DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

ONDERZOEKSRAPPORT NR 9613

A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations

by

Bert DE REYCK

Willy HERROELEN

Katholieke Universiteit Leuven

Naamsestraat 69, B-3000 Leuven

ONDERZOEKSRAPPORT NR 9613

A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations

by

Bert DE REYCK
Willy HERROELEN

D/1996/2376/13

A BRANCH-AND-BOUND PROCEDURE FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM WITH GENERALIZED PRECEDENCE RELATIONS

Bert DE REYCK

Willy HERROELEN

Operations Management Group
Department of Applied Economics
Katholieke Universiteit Leuven
Hogenheuvel College
Naamsestraat 69, B-3000 Leuven, Belgium
Phone: 32-16-32 69 66 or 32-16-32 69 69 70
Fax: 32-16-32 67 32
E-mail: Bert.DeReyck@econ.kuleuven.ac.be or Willy.Herroelen@econ.kuleuven.ac.be
WWW-page: http://econ.kuleuven.ac.be/tew/academic/om/people/bert
http://econ.kuleuven.ac.be/tew/academic/om/people/willy

A BRANCH-AND-BOUND PROCEDURE FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM WITH GENERALIZED PRECEDENCE RELATIONS

Bert De Reyck • Willy Herroelen

Departement Toegepaste Economische Wetenschappen, Katholieke Universiteit Leuven

ABSTRACT

We present an optimal procedure for the resource-constrained project scheduling problem (RCPSP) with generalized precedence relations (further denoted as RCPSP-GPR) with the objective of minimizing the project makespan. The RCPSP-GPR extends the RCPSP to arbitrary minimal and maximal time lags between the starting and completion times of activities. The procedure is a depth-first branch-and-bound algorithm in which the nodes in the search tree represent the original project network extended with extra precedence relations which resolve a resource conflict present in the parent node. Resource conflicts are resolved using the concept of minimal delaying alternatives, i.e. minimal sets of activities which, when delayed, release enough resources to resolve the conflict. Precedence- and resource-based lower bounds as well as dominance rules are used to fathom large portions of the search tree. The procedure can be extended to other regular measures of performance by some minor modifications. Even non-regular measures of performance, such as the maximization of the net present value of the project or resource levelling objectives, can be handled. The procedure has been programmed in Microsoft® Visual C++ for use on a personal computer. Extensive computational experience is obtained.

KEYWORDS

Project scheduling; Resource constraints; Generalized precedence relations; Branch-and-bound

1. Introduction

CPM (Critical Path Method; Kelley and Walker, 1959) and PERT (Program Evaluation and Review Technique; Malcolm et al., 1959) are basically devoted to project scheduling under the assumption that required resources are available in sufficient amounts, and that the technological precedence relations between any pair of activities i and j imply *strict* precedence, meaning that activity i must be completed before activity j can be initiated. For many years, the assumption of sufficiently available resources has been relaxed and many research efforts have been directed towards project scheduling with explicit consideration of resource requirements and constraints. Davis (1973) categorized these models into three classes: time/cost trade-off problems, resource-constrained project scheduling problems and resource levelling problems.

More recent research has been directed at relaxing the strict precedence assumption of CPM/PERT. The resulting types of precedence relations are often referred to as MPM (Metra Potential Method) precedence constraints (Kerbosh and Schell, 1975; Zhan, 1994), precedence diagramming relations (Moder et al., 1983), time windows (Bartusch et al., 1988), minimal and maximal time lags (Brinkmann and Neumann, 1994; Neumann and Schwindt, 1995; Schwindt, 1995; Neumann and Zhan, 1996), and generalized precedence constraints (Wikum et al., 1994). In accordance with Elmaghraby and Kamburowski (1992), we denote them as *generalized precedence relations* (*GPRs*). We distinguish between four types of GPRs: start-start (SS), start-finish (SF), finish-start (FS) and finish-finish (FF).

GPRs can specify a minimal or maximal time lag between any pair of activities. A minimal time lag specifies that an activity can only start (finish) when the predecessor activity has already started (finished) for a certain time period. A maximal time lag specifies that an activity should be started (finished) at the latest a certain number of time periods beyond the start (finish) of another activity. Many specific situations can be readily modelled using GPRs, such as (Bartusch et al., 1988; De Reyck, 1995b; Neumann and Schwindt, 1995):

- activity ready times (release dates) and deadlines
- activities that have to start (terminate) simultaneously
- activities that have to terminate (can only start) x time units before the project completion
- non-delay execution of (precedence related or unrelated) activities
- total overlapping / strong partial overlapping / weak partial overlapping of activities
- fixed activity starting times
- time-varying resource-requirements and / or resource availabilities
- time-windows for resources
- inventory (work in process) restrictions
- setup times, overlapping production activities (process batches, transfer batches)
- assembly line zoning constraints

The first treatment of GPRs is due to Kerbosch and Schell (1975), based on the pioneering work of Roy (1962). Other studies include Crandall (1973), Elmaghraby (1977), Wiest (1981), Moder et al. (1983), Bartusch et al. (1988), Elmaghraby and Kamburowski (1992), Brinkmann and Neumann (1994), Zhan (1994), De Reyck (1995a, 1995b), Neumann and Schwindt (1995) and Schwindt (1995) and Neumann and Zhan (1996). Demeulemeester and Herroelen (1996a) have extended their branch-and-bound procedure for the RCPSP to the case of minimal time lags, activity release dates and deadlines and variable resource availabilities. To the best of our knowledge, the only optimal solution procedure reported in the literature for the RCPSP-GPR is the branch-and-bound procedure of Bartusch et al. (1988). Heuristic solution procedures are provided by Brinkmann and Neumann (1994), Zhan (1994) and Neumann and Zhan (1996). In this paper we present a new branch-and-bound procedure for the RCPSP-GPR supported by extensive computational tests.

The remainder of this paper is organized as follows. Section 2 elaborates on the concept of GPRs. Section 3 continues with the temporal analysis of activity networks with GPRs. In section 4 , which discusses the resource analysis of such networks, a branch-and-bound procedure for the RCPSP-GPR is presented. Computational results are given in section 5. Section 6 is reserved for our overall conclusions and suggestions for future research.

2. Generalized precedence relations (GPRs)

The extension of the Critical Path Method (*CPM*) to networks with GPRs was originally called the Metra-Potential Method (*MPM;* Kerbosh and Schell, 1975). A basic assumption of CPM is that the precedence relations between the activities are of the finish-start type (with a time lag of zero). They imply a strict precedence because the predecessor activity must be completed before the successor activity can be initiated. These CPM networks can be represented by an acyclic activity-on-arc network, as in the original work of Kelley and Walker (1959), or by an acyclic activity-on-node network, which has gained more popularity.

CPM can easily be extended to GPRs, but only in the case of minimal time lags between activities, or, more correctly, only in the special case in which there are no arcs with negative length in the constraint digraph (cf. infra) and, consequently, no cycles of precedence relations. The same applies to the resource-constrained project scheduling problem (RCPSP), which can easily be extended to cope with minimal time lags (Demeulemeester and Herroelen 1996a). Activity networks with GPRs, however, can deal with activities among which maximal, as well as minimal time lags exist. The precedence relations specifying a maximal time lag can be represented by a negative minimal time lag in the opposite direction. Consequently, networks that include activities among which minimal and maximal time lags exist can be represented as

cyclic networks. For instance, a maximal finish-start lag of 5 between activity i and j is equivalent to a minimal start-finish lag of -5 between activity j and i, as is shown in Fig. 1.

Figure 1. The equivalence of maximal and minimal time lags

Assume a project represented in activity-on-node (AoN) notation by a directed graph $G = \{V, E\}$ in which V is the set of vertices or activities, and E is the set of edges or GPRs. The non-preemptable activities are numbered from 1 to n, where the dummy activities 1 and n mark the beginning and the end of the project. The duration of an activity is given by $d_i (1 \le i \le n)$, its starting time by $s_i (1 \le i \le n)$ and its finishing time by $f_i (1 \le i \le n)$. There are m *renewable* resource types, with r_{ikx} $(1 \le i \le n, 1 \le k \le m, 1 \le x \le d_i)$ the resource requirements of activity i with respect to resource type k in the x^{th} period it is in progress and $a_{kt} (1 \le k \le m; 1 \le t \le T)$ the availability of resource type k in time period $]t\text{-}1, t]$ (T is an upper bound on the project length). If the resource requirements and availabilities are not time-dependent, they are represented by r_{ik} $(1 \le i \le n, 1 \le k \le m)$ and $a_k (1 \le k \le m)$ respectively. The minimal and maximal time lags between two activities i and j have the form:

$$s_i + SS_{ij}^{\min} \le s_j \le s_i + SS_{ij}^{\max}$$

$$s_i + SF_{ij}^{\min} \le f_j \le s_i + SF_{ij}^{\max}$$

$$f_i + FS_{ij}^{\min} \le s_j \le f_i + FS_{ij}^{\max}$$

$$f_i + FF_{ij}^{\min} \le f_j \le f_i + FF_{ij}^{\max}$$

The different types of GPRs can be represented in a *standardized form* by reducing them to just one type, e.g. the minimal start-start precedence relations, using the following transformation rules (Bartusch et al., 1988):

$$\begin{aligned}
s_i + SS_{ij}^{\min} \le s_j &\Rightarrow s_i + l_{ij} \le s_j \quad \text{with} \quad l_{ij} = SS_{ij}^{\min} \\
s_i + SS_{ij}^{\max} \ge s_j &\Rightarrow s_j + l_{ji} \le s_i \quad \text{with} \quad l_{ji} = -SS_{ij}^{\max} \\
s_i + SF_{ij}^{\min} \le f_j &\Rightarrow s_i + l_{ij} \le s_j \quad \text{with} \quad l_{ij} = SF_{ij}^{\min} - d_j \\
s_i + SF_{ij}^{\max} \ge f_j &\Rightarrow s_j + l_{ji} \le s_i \quad \text{with} \quad l_{ji} = d_j - SF_{ij}^{\max} \\
f_i + FS_{ij}^{\min} \le s_j &\Rightarrow s_i + l_{ij} \le s_j \quad \text{with} \quad l_{ij} = d_i + FS_{ij}^{\min} \\
f_i + FS_{ij}^{\max} \ge s_j &\Rightarrow s_j + l_{ji} \le s_i \quad \text{with} \quad l_{ji} = -d_i - FS_{ij}^{\max} \\
f_i + FF_{ij}^{\min} \le f_j &\Rightarrow s_i + l_{ij} \le s_j \quad \text{with} \quad l_{ij} = d_i - d_j + FF_{ij}^{\min} \\
f_i + FF_{ij}^{\max} \ge f_j &\Rightarrow s_j + l_{ji} \le s_i \quad \text{with} \quad l_{ji} = d_j - d_i - FF_{ij}^{\max}
\end{aligned}$$

If there is more than one time lag l_{ij} between two activities i and j, only the maximal time lag is retained. The interval $[s_i + l_{ij}\ ,\ s_i - l_{ij}]$ is called the *time window* of s_j relative to s_i (Bartusch et al., 1988). Applying these transformation rules to an activity network with GPRs results in a so-called *constraint digraph*, which is short for *digraph of temporal constraints* (Bartusch et al., 1988). An example of such a constraint digraph is given in Fig. 2. The labels above the nodes denote the activity durations d_i. The labels associated with the arrows indicate the time lags l_{ij}. The constraint digraph contains less information than the original activity network. For instance, the effect of an increase or decrease in activity durations cannot be examined correctly in the constraint digraph. This, however, only poses a problem when the activity duration is subject to change as in time/cost trade-off problems or in multi-mode problems.

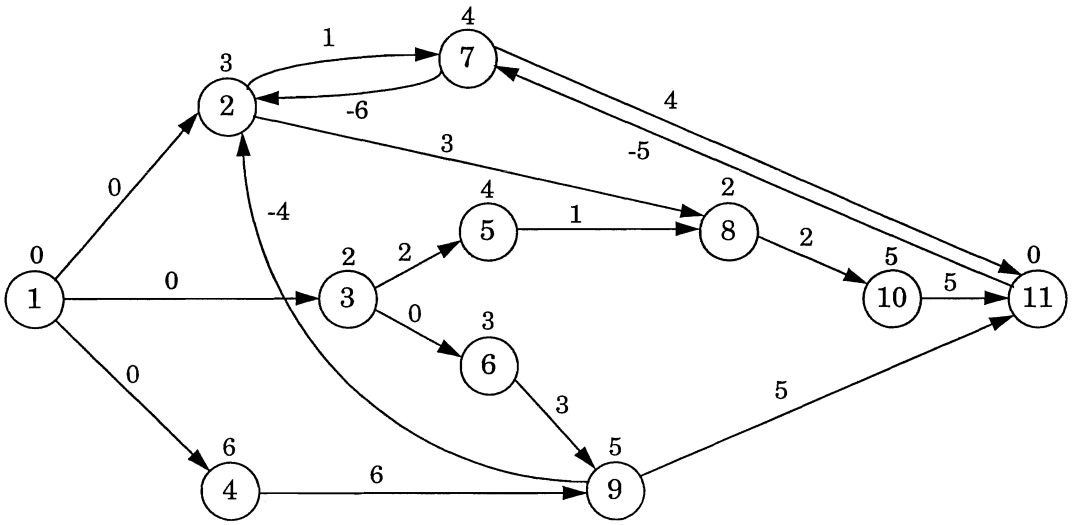

Figure 2. Constraint digraph of an activity network with GPRs

Because activity networks with GPRs contain cycles, additional concepts are needed (Bartusch et al., 1988). A path $<i_s, i_k, i_l, ..., i_t>$ is called a *cycle* if $s = t$. With 'path' we mean a *directed* path, and with 'cycle' we mean a *directed* cycle. The *length* of a path (cycle) is defined as the sum of all the lags associated with the arcs belonging to that path (cycle). Activity durations do not have to be included in the calculation of a path length, since all time lags l_{ij} in a constraint digraph are of the SS-type. To ensure that the dummy start and finish activities correspond to the beginning and the completion of the project, we assume that there exists at least one path with nonnegative length from node 1 to every other node and at least one path from every node i to node n which is equal to or larger than d_i. If there are no such paths, we can insert arcs $(1,i)$ or (i,n) with weight zero or d_i respectively. In the example in Fig. 2, such arcs were added between node 1 and nodes 2, 3 and 4, and between nodes 7, 9 and 10 and node 11. $P(i) = \{j \mid (j,i) \in E\}$ is

the set of all *immediate predecessors* of node i, $Q(i) = \{j \mid (i,j) \in E\}$ is the set of all its *immediate successors*. If there exists a path from i to j, then we call i a *predecessor* of j and j a *successor* of i. $P^*(i)$ and $Q^*(i)$ denote the set of predecessors and successors of node i respectively. If the length of the longest path from i to j is positive or all arcs of a longest path are associated with a lag of zero, node i is called a *real* (immediate) predecessor of node j, and j is called a *real* (immediate) successor of i. Otherwise it is a *fictitious* one.

3. Temporal analysis

The goal of project scheduling problems is to obtain a *schedule* S, which is a vector of starting times $\{s_1, s_2, ..., s_n\}$ for all activities. Schedules are subject to temporal constraints and resource constraints. In this section, we focus on the temporal constraints. In section 4, additional resource constraints will be taken into account. A schedule is called *time-feasible*, if all the starting times satisfy all (generalized) precedence relations. In other words, a time-feasible schedule with starting times $\{s_1, s_2, ..., s_n\}$ satisfies the conditions that:

$$\begin{cases} s_i \geq 0 & \forall\, i \in V \quad [1] \\ s_i + l_{ij} \leq s_j & \forall\, (i,j) \in E \quad [2] \end{cases}$$

where Eqs. 1 ensure that no activity starts before the current time (time zero), and Eqs. 2 denote the precedence constraints in standardized form. Notice that Eqs. 1 have to be included, not only for the starting activities, but for every activity in the network, since the time lags l_{ij} can assume negative values. The minimum starting times $\{s_1, s_2, ..., s_n\}$ satisfying both Eqs. 1 and 2 form the *early start schedule* $ESS = \{es_1, es_2, ..., es_n\}$ associated with the temporal constraints. For the example, ESS = {0, 2, 0, 0, 2, 0, 7, 5, 6, 7, 12}, as represented by the Gantt-chart in Fig. 3. The arrows indicate the time lags that are binding. The dotted line represents the end of the project (finishing time of the dummy end activity).

The calculation of an ESS can be related to the test for existence of a time-feasible schedule. The earliest start of an activity i can be calculated by finding the longest path from node 1 to node i. We also know that there exists a time-feasible schedule for G iff G has no cycle of positive length (Bartusch et al., 1988). Cycles of positive length would unable us to calculate starting times for the activities which satisfy conditions [1] and [2]. Therefore if we calculate the distance matrix $D = [d_{ij}]$, where d_{ij} denotes the maximal distance (path length) from node i to node j, a positive path length from node i to itself indicates the existence of a cycle of positive length and, consequently, the non-existence of a time-feasible schedule.

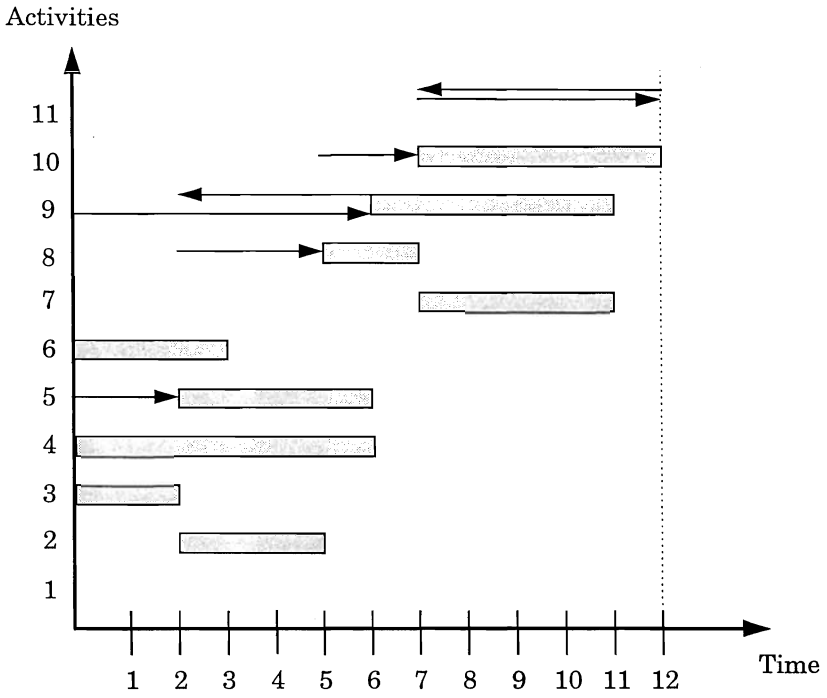

Figure 3. A Gantt-chart of the *ESS*

The calculation of the distance matrix D can be done by standard graph algorithms for longest paths in networks, for instance by the Floyd-Warshall algorithm (for details, see Lawler, 1976). If we start with the matrix $D^{(1)} = \left[d_{ij}^{(1)}\right]$ $(i, j = 1, 2, ..., n)$ with

$$d_{ij}^{(1)} = \begin{cases} 0 & \text{if } i = j \\ l_{ij} & \forall\, (i, j) \in E \\ -\infty & \text{otherwise} \end{cases}$$

we can compute the matrix $D = D^{(n+1)}$ according to the updating formula $d_{ij}^{(v)} = \max\left\{d_{ij}^{(v-1)}, d_{il}^{(v-1)} + d_{lj}^{(v-1)}\right\}$ $(i, j, l = 1, 2, ..., n)$. If $d_{ii} = 0$ for all $i = 1, 2, ..., n$ (the numbers in the diagonal of D), there exists a time-feasible schedule. The *ESS* is given by the numbers in the upper row of D: $ESS = \{d_{1,1}, d_{1,2}, ..., d_{1,n}\}$.

The computation of D takes $O(|V|^3)$ time (Bartusch et al., 1988). The *ESS* can be calculated more efficiently by using the Modified Label Correcting Algorithm (Ahuja et al., 1989), which is of time complexity $O(|V|\,|E|)$ and which also allows for the identification of positive cycles and for the calculation of a *late start schedule* $LSS = \{ls_1, ls_2, ..., ls_n\}$. The *existence* of positive cycles can also be verified by employing the modified Bellman algorithm of time complexity $O(|V|\,|E|)$, and the *identification* of such cycles can be accomplished by the algorithm of Jensen and Barnes (1987), which is also of time complexity $O(|V|\,|E|)$.

4. Resource analysis

The resource-constrained project scheduling problem with generalized precedence relations (RCPSP-GPR) can be conceptually formulated as follows:

$$\text{Minimize } s_n \qquad [3]$$

Subject to

$$s_i + l_{ij} \le s_j \qquad \forall (i,j) \in E \qquad [4]$$

$$\sum_{i \in S(t)} r_{ik} \le a_{kt} \qquad k = 1,2,\ldots,m \qquad t = 1,2,\ldots,T \qquad [5]$$

$$s_1 = 0 \qquad [6]$$

$$s_i \in \mathrm{N} \qquad i = 1,2,\ldots,n \qquad [7]$$

No splitting allowed

where $S(t)$ is the set of activities in progress in time period $]t\text{-}1, t]$ and T is an upper bound on the project duration, for instance $T = \sum_{i \in V} \max\left\{d_i, \max_{j \in Q(i)}\left\{l_{ij}\right\}\right\}$ (for the example in Fig. 2, $T = 34$). Note that it is not always possible to derive a feasible solution. The upper bound T indicates the maximal value for the project makespan if a feasible solution exists. The objective function given in Eq. 3 minimizes the project duration, given by the starting time (or finishing time, since $d_n = 0$) of the dummy activity n. The precedence constraints are denoted in standardized form by Eqs. 4. Eqs. 5 represent the resource constraints. The resource requirements and availabilities are assumed to be constant over time, although this assumption can be easily relaxed using GPRs without having to change the solution procedures (Bartusch et al., 1988). Eq. 6 forces the dummy start activity to begin at time zero and Eqs. 7 ensure that the activity starting times assume nonnegative integer values. Once started, activities run to completion. However, this nonpreemption condition can easily be relaxed by splitting up the activities in unit-duration *subactivities* (Demeulemeester and Herroelen, 1996b), connected with strict precedence relations (zero-lag finish-start precedence relations).

The RCPSP-GPR is known to be strongly NP-hard, and even the decision problem of testing whether a RCPSP-GPR instance has a feasible solution is NP-complete (Bartusch et al., 1988). Demeulemeester and Herroelen (1996a) developed a branch-and-bound procedure for the *generalized resource-constrained project scheduling problem (GRCPSP)* which allows for minimal time lags only, with the additional assumption that a successor activity can never start before its predecessor (i.e. no negative lags in the constraint digraph). To the best of our knowledge, the only optimal solution procedure presented in the literature for the RCPSP-GPR is the branch-and-bound algorithm of Bartusch et al. (1988). In this section, we discuss a new branch-and-bound procedure for the RCPSP-GPR based on the concepts of minimal delaying alternatives as developed by Demeulemeester and Herroelen (1992) for the RCPSP and adapted by Icmeli and

Erengüç (1996) for the RCPSP with discounted cash flows. The procedure uses several lower bounds, including a generalized version of a lower bound for the RCPSP proposed by Mingozzi et al. (1994) and several powerful dominance rules.

4.1. The search tree

The nodes in the search tree represent the initial project network, described by a distance matrix $D=[d_{ij}]$, extended with extra (zero-lag finish-start) precedence relations to resolve a resource conflict present in the parent node, which results in an *extended* distance matrix. Nodes which represent time-feasible (no violated maximal time lags) but resource-infeasible project networks and which are not fathomed by any node fathoming rules described below lead to a new branching. Therefore each (undominated) node represents a time-feasible, but not necessarily resource-feasible project network. A similar branching scheme is used by Bartusch et al. (1988). However, these authors use the concept of a *reduced forbidden set* (a minimal set of activities which cannot be scheduled together within the resource constraints) to resolve the resource conflicts. Resource conflicts are then resolved by successively adding precedence relations such that each forbidden set is no longer scheduled in parallel. Each possible combination of added precedence relations leads to a new node. A similar branching scheme as the one of Bartusch et al. (1988) was used by Bell and Park (1990) for the RCPSP. Bell and Park (1990) use the concept of a *minimal resource violating set*, which is equivalent to a reduced forbidden set.

In our procedure, resource conflicts are resolved using the concept of *minimal delaying alternatives* (Demeulemeester and Herroelen, 1992), i.e. minimal sets of activities which, when delayed, release enough resources to resolve the resource conflict (and which do not contain any other delaying alternative as a subset). Each of these minimal delaying alternatives is then delayed (enforced by extra strict precedence relations $i \prec j$, implying $s_i + d_i \leq s_j$) by each of the activities also belonging to the *conflict set* $S(t^*)$, the set of activities in progress in period $]t^*\text{-}1, t^*]$ (the period of the *first* resource conflict), but not belonging to the delaying alternative.

A similar delaying strategy was used by Demeulemeester and Herroelen (1992) for the RCPSP. As the RCPSP can be solved using semi-active timetabling to construct the partial schedules, activities belonging to the minimal delaying alternative can be delayed by the activity in $S(t^*)$ which terminates at the earliest time instant after the current decision point. In the RCPSP-GPR, this delaying strategy cannot be used because of the presence of maximal time lags, which make semi-active timetabling inappropriate. The same problem occurs in the RCPSP with discounted cash flows (RCPSP-DC). In the RCPSP-DC, semi-active timetabling is also inappropriate and a modified delaying scheme has to be used. Icmeli and Erengüç (1996) have

modified the delaying scheme of Demeulemeester and Herroelen (1992) for the RCPSP-DC. A similar scheme can be used for the RCPSP-GPR.

There are several possible *delaying modes* for delaying a delaying alternative. In the procedure of Demeulemeester and Herroelen (1992), no distinction was needed between minimal delaying alternatives and minimal delaying modes because there was a one-to-one correspondence between the two. In the RCPSP-GPR, one delaying alternative can give rise to several delaying modes, possibly one for each activity in $S(t^*)$ which is not an element of the delaying alternative.

Assume, for example, that in a certain period $]t^*\text{-}1,\ t^*]$, 4 activities are in progress and cause a resource conflict: $S(t^*) = \{1,2,3,4\}$. Suppose that the minimal delaying alternatives are {1}, {2} and {3, 4}, i.e. delaying activity 1, activity 2 or activities 3 and 4 simultaneously releases enough resources to resolve the resource conflict. For the RCPSP, using the procedure of Demeulemeester and Herroelen (1992), we would create 3 new nodes (three minimal delaying modes). In the first node, activity 1 is delayed by the earliest finishing activity (x) among activities 2, 3 or 4 ($x \prec 1$). In the second node, activity 2 is delayed by the earliest finishing activity (y) among activities 1, 3 or 4 ($y \prec 2$). Finally, activities 3 and 4 are delayed by activity 1 or 2 (z), depending on which activity finishes the earliest ($z \prec 3$ and $z \prec 4$). This results in 3 new nodes, as illustrated in Fig. 4.

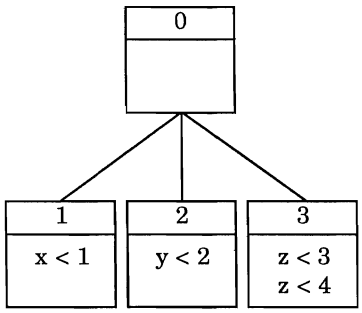

Figure 4. Delaying strategy for the RCPSP of Demeulemeester and Herroelen (1992)

For the RCPSP-GPR, as for the RCPSP-DC (Icmeli and Erengüç, 1996), the delay of activity 1 is established by adding a precedence relation between activity 2, 3 or 4 and activity 1. We therefore create three new nodes (instead of one), one with the precedence relation $2 \prec 1$, one with the precedence relation $3 \prec 1$ and one with the precedence relation $4 \prec 1$. Delaying activity 2 is accomplished by creating 3 new nodes with the extra precedence relations $1 \prec 2$, $3 \prec 2$ and $4 \prec 2$. Delaying activities 3 and 4 is accomplished by creating two new nodes with the extra precedence relations $1 \prec 3$ and $1 \prec 4$, or $2 \prec 3$ and $2 \prec 4$. In total, 8 new nodes (minimal delaying modes) are created, as illustrated in Fig. 5.

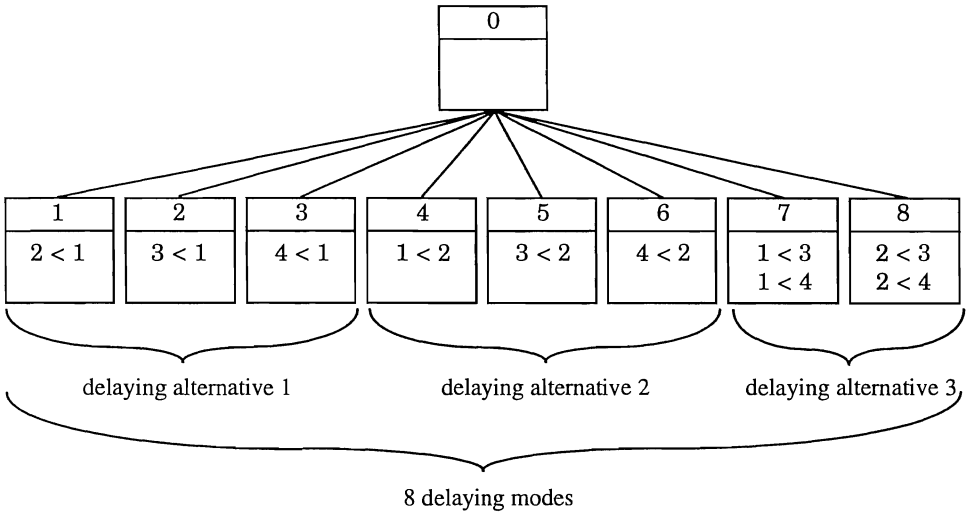

Figure 5. Delaying strategy for the RCPSP-GPR

In general, the *delaying set D*, i.e. the set of all minimal delaying alternatives, is equal to $D = \left\{ D_d \middle| D_d \subset S(t^*) \text{ and } \forall \text{ resource type } k : \sum_{i \in S(t^*)} r_{ik} - \sum_{i \in D_d} r_{ik} \le a_k \text{ and } \forall D_{d'} \in \mathrm{D} : D_{d'} \not\subset D_d \right\}$. The set of minimal delaying modes equals: $M = \left\{ M_m \middle| M_m = \{k \prec D_d\}, k \in S(t^*) \setminus D_d, D_d \in D \right\}$. Activity k is called the *delaying activity*: $k \prec D_d$ implies that $k \prec l$ for all $l \in D_d$.

Each minimal delaying mode is then examined for time-feasibility and, if time-feasible, evaluated by computing the critical path based lower bound lb_0. Each time-feasible minimal delaying mode with a lower bound $lb_0 \le T$ is then considered for further branching, and branching occurs from the node with the smallest lower bound lb_0. If the node represents a project network in which a resource conflict occurs, a new branching occurs. If it represents a feasible schedule, the lower bound T is updated and the procedure backtracks to the previous level in the search tree. Therefore, we have a depth-first search procedure, in which branching occurs until at a certain level in the tree, there are no delaying modes left to branch from. Then, the procedure backtracks to the previous level in the search tree and reconsiders the other delaying modes (not yet branched from) at that level. The procedure stops when it backtracks to level 0.

THEOREM 1. The delaying strategy which consists of delaying all minimal delaying alternatives D_d by each activity $k \in S(t^*) \setminus D_d$ will lead to the optimal solution of the RCPSP-GPR in a finite number of steps.

PROOF. See Appendix.

4.2. Node fathoming rules

Nodes are fathomed when they represent a time-infeasible project network or when lb_0 exceeds T. Nodes which are not fathomed and still represent an infeasible project network are considered for further branching. Four other node fathoming rules are added, three dominance rules and a lower bound rule. We also add a procedure which reduces the solution space.

4.2.1. Redundant delaying alternatives

Because activity overlaps are allowed ($d_{ij} < d_i$), it is possible that in period $]t^*\text{-}1, t^*]$ (the period of the first resource conflict), the set of activities in progress (the conflict set $S(t^*)$) contains an activity i together with a *real* successor j of activity i ($d_{ij} \geq 0$). Then, delaying activity i will inevitably also delay activity j. Therefore, we can extend each minimal delaying alternative D_d with the real successors j ($j \notin D_d$) of an activity $i \in D_d$. As a result of this operation, minimal delaying alternatives may become non-minimal, which can be eliminated from further consideration.

THEOREM 2. If there exists a minimal delaying alternative D_d with activity $i \in D_d$ but its real successor $j \notin D_d$ ($d_{ij} \geq 0$), we can extend D_d with activity j. Any minimal delaying alternative becoming non-minimal as a result of this operation may be eliminated from further consideration.
PROOF. Obvious.

4.2.2. Redundant delaying modes

Again, because of the possibility of activity overlaps, it is possible that a certain minimal delaying alternative D_d is giving rise to two delaying modes M_{m_1} and M_{m_2}, in which the delaying activities i ($i \prec D_d$) and j ($j \prec D_d$) are precedence related. If $d_{ij} + d_j \geq d_i$, that is, in any feasible solution, activity j will terminate after activity i, we can eliminate delaying mode M_{m_2} from further consideration because it will never lead to a superior solution than delaying mode M_{m_1}.

THEOREM 3. When a minimal delaying alternative D_d gives rise to two delaying modes M_{m_1} and M_{m_2} with delaying activities i and j respectively, *delaying mode M_{m_2} is dominated by delaying mode M_{m_1} iff $d_{ij} + d_j \geq d_i$.*
PROOF. Obvious.

4.2.3. A time- and resource-based lower bound

Recently, Mingozzi et al. (1994) have developed five new lower bounds, lb_1, lb_2, lb_p, lb_x and lb_3, derived from different relaxations of a new mathematical formulation for the RCPSP. Some of these new lower bounds (namely lb_1, lb_2, lb_x and lb_3) dominate the critical path based lower bound (lb_0) and they all prove to be tighter than the critical sequence lower bound (lb_s) of Stinson et al. (1978) on the 110 RCPSP instances assembled by Patterson (1984) and the 480 randomly generated RCPSP instances of Kolisch et al. (1992). Mingozzi et al. (1994) have also developed a new branch-and-bound procedure for the RCPSP based on this new mathematical formulation, which incorporates the most promising new lower bound lb_3.

Mingozzi et al. (1994) compute lb_3 using a heuristic for the weighted node packing problem. Demeulemeester and Herroelen (1995) have incorporated another version of lb_3 (lb_3') in their procedure for the RCPSP: For each activity $i \in V$, its possible *companions*, i.e. the activities with which it can be scheduled in parallel, respecting both the precedence and resource constraints, are determined. All (unscheduled) activities i are then entered in a list L in non-decreasing order of the number of companions (non-increasing duration as tie-breaker). The following procedure then yields a lower bound, lb_3' (for the partial schedule under consideration):

$lb_3' = 0$ (or the earliest completion time of the activities in progress if a partial schedule is already determined)
while L not empty **do**
    take the first activity (activity i) in L
    $lb_3' = lb_3' + d_i$
    remove activity i and its companions from L
enddo

Computational results obtained by Demeulemeester and Herroelen (1995) indicate that lb_3' indeed outperforms lb_0 and that incorporating lb_3' in their branch-and-bound procedure reduces the computational effort to solve the 110 problems of Patterson (1984) and the 480 problems of Kolisch et al. (1992). Note that, contrary to lb_3 as calculated by Mingozzi et al. (1994), the procedure above may yield an lb_3' which is smaller than lb_0, but this constitutes no real disadvantage since lb_0 is calculated as well and the final value for the lower bound equals $lb = \max\{lb_0, lb_3'\}$.

The procedure of Demeulemeester and Herroelen (1995) for computing lb_3' can be extended to the RCPSP-GPR, by changing the calculation of the companions of the activities. In the RCPSP-GPR, two activities i and j between which a precedence relation exists, are companions if the resource requirements of both activities do not exceed the resource availability for any resource type, and if both $d_{ij} < d_i$ and $d_{ji} < d_j$.

In our implementation of lb_3, we have also adapted the (weighted node packing) heuristic. Instead of removing an activity j from the list L when a companion i is taken from the list, we only remove part of activity j from the list. The logic behind this reasoning relies on both a *duration* and *time lag argument*. The *duration argument* goes as follows: When an activity i is scheduled, a companion j can be scheduled in parallel with i. However, if $d_i < d_j$, only a part of activity j can be scheduled in parallel with i. Therefore, a part of activity j (with remaining duration $d_j^r = d_j - d_i$) can be left in L. Initially, all d_j^r are equal to d_j.

It is clear that lb_3, even using the duration argument as described above, will not perform as well as lb_3' did for the RCPSP without GPRs. The reason for this is that activities will have many more companions in the RCPSP-GPR than in the RCPSP. Even when activities i and j can overlap for only one time unit (because of a minimal start-start time lag equal to $d_i - 1$), they will be considered as companions. Therefore, when using lb_3 for the RCPSP-GPR in an effective way, we will have to look at the time lags between the activities, leading to our *time lag argument*: We adjust the part of activity j which has to be removed when a companion i is taken from the list, by incorporating the precedence relations between i and j. Suppose, for instance, that two companion-activities, i and j, are in L (d_i = 3, d_j = 5 en l_{ij} = 2). Using the heuristic of Demeulemeester and Herroelen (1995), we would remove activity j from the list when activity i is taken from the list, which leads to an increase in lb_3' by d_i. Using our duration argument made above, we would leave activity j in the list, but only with a duration d_j^r equal to d_j - d_i = 2. If no other companion of j is taken from the list, lb_3 could be as much as two units higher than in the case where activity j would be completely removed from L. Taking into account the start-start precedence relation l_{ij} = 2 between i and j, we see that activity i and j can only overlap for one time unit. Therefore, using our time lag argument, only one time unit has to be subtracted from d_j, possibly leading to a further increase in lb_3.

However, several different types of activity overlaps may occur. Therefore, we have to look at each combination of minimal and maximal time lags between two activities i and j when

deciding how much to remove from activity j. Table I shows the appropriate action (namely how much to remove from activity j) for each combination. Note that when we want to remove x units from activity j whereas only y ($y<x$) units are left, activity j is to be removed completely from L.

Table I. The calculation of d_j^r

	$d_i \le d_{ij}$	$0 < d_{ij} < d_i$	$-d_j < d_{ij} \le 0$	$d_{ij} \le -d_j$
$d_j \le d_{ji}$	infeasible	infeasible	infeasible	no companions
$0 < d_{ji} < d_j$	infeasible	infeasible	$d_j^r = d_j^r - \min\{d_j - d_{ji}, d_i\}$	$d_j^r = d_j^r - \min\{d_j - d_{ji}, d_i\}$
$-d_i < d_{ji} \le 0$	infeasible	$d_j^r = d_j^r - (d_i - d_{ij})$	$d_j^r = d_j^r - d_i$	$d_j^r = d_j^r - d_i$
$d_{ji} \le -d_i$	no companions	$d_j^r = d_j^r - (d_i - d_{ij})$	$d_j^r = d_j^r - d_i$	$d_j^r = d_j^r - d_i$

Because in an optimal solution procedure for the RCPSP-GPR, time-infeasibilities will be detected before lb_3 is calculated, the calculation of lb_3 for the RCPSP-GPR (lb_3^g) can now be summarized as follows:

$lb_3^g = 0$
while L not empty **do**
  take the first activity (activity i) in L and remove it from L
  $lb_3^g = lb_3^g + d_i^r$
  for every companion j of i **do**
    if $l_{ij} > 0$ **then** $d_j^r = d_j^r - (d_i - d_{ij})$
    else if $l_{ji} > 0$ **then** $d_j^r = d_j^r - \min\{d_j - d_{ji}, d_i\}$
    else $d_j^r = d_j^r - d_i$
    endif
  if $d_j^r \le 0$, remove activity j from L
  enddo
enddo

THEOREM 4. *lb_3^g is a valid lower bound for the RCPSP-GPR.*

PROOF. See appendix.

lb_3^g is used to fathom nodes for which $lb_3^g \geq T$. However, whereas lb_0 is calculated immediately upon the creation of a node, the calculation of lb_3^g is deferred until a decision has been made to actually branch from that node. The rationale behind this is that (a) lb_3^g is much more difficult to compute than lb_0, and that (b) calculating lb_3^g implies calculating the entire distance matrix (time complexity $O[n^2]$ using the algorithm described in section 4.3). Supported by extensive computational tests, we defer the calculation of lb_3^g and the distance matrix until the node is actually selected for branching. As a result, only lb_0 is used as a branching criterion. Time-infeasibilities (which lead to node fathoming) or feasible solutions (which lead to an upper bound update) are also detected only when that node is selected to branch from.

4.2.4. A subset dominance rule

The branch-and-bound procedure starts with the early start schedule for the project network in the root node of the search tree, and successively adds precedence constraints in order to eliminate resource conflicts. Each node represents the initial project network extended with a set of (strict) precedence constraints. Therefore, it is possible that a certain node represents a project network which has been examined earlier at another node in the search tree, in which case the corresponding distance matrices will be identical. Therefore, it suffices to compare these distance matrices. However, saving and comparing distance matrices would lead to an enormous increase in memory requirements and computational effort. Another way of checking whether two nodes represent the same project network, is to check the added precedence constraints. Identical sets of precedence constraints lead to identical project networks. Examining the sets of precedence constraints also allows us to check whether a set of precedence constraints in a node contains as a subset another set of precedence constraints obtained in another node, in which case the former node can be eliminated from further consideration, since it can never lead to a superior solution.

THEOREM 5. If the set of added precedence constraints which leads to the project network (in the form of an extended distance matrix) in node x contains as a subset another set of precedence constraints leading to the project network (extended distance matrix) in a previously examined node y in another branch of the search tree, node x can be fathomed.
PROOF. See appendix.

Notice, however, that the set of added precedence constraints in a certain node always contains as a subset the set of precedence constraints in its parent node (or grandparent node, ...). Obviously, this rule only applies when a set of precedence constraints is compared to another set of precedence constraints obtained *along another path of the search tree*. This can easily be enforced by only saving information for node dominance testing during backtracking.

The question remains which nodes exactly have to be saved in order to be able to test this dominance rule. When a node x is dominated by a node y, it will also be dominated by a node z which is the parent node of node y unless it is also a parent node of node x. Therefore, to check whether node x is dominated, we save the set of added precedence constraints of the nodes which are no parent node of x, but for which the parent node is a parent node of x. Suppose that for the example search tree in Fig. 6, all nodes on a certain level are sorted from left to right in nondecreasing order of lb_0 (which is the order in which they will be selected to branch from). In order to examine whether node x is dominated by another node examined earlier, we only have to examine all nodes displayed in bold. All other nodes are either a parent node of node x, or contain no useful information since a parent node can equally well dominate node x. When we reach node x, only the information concerning the four bold nodes has to be saved in order to be able to test node dominance.

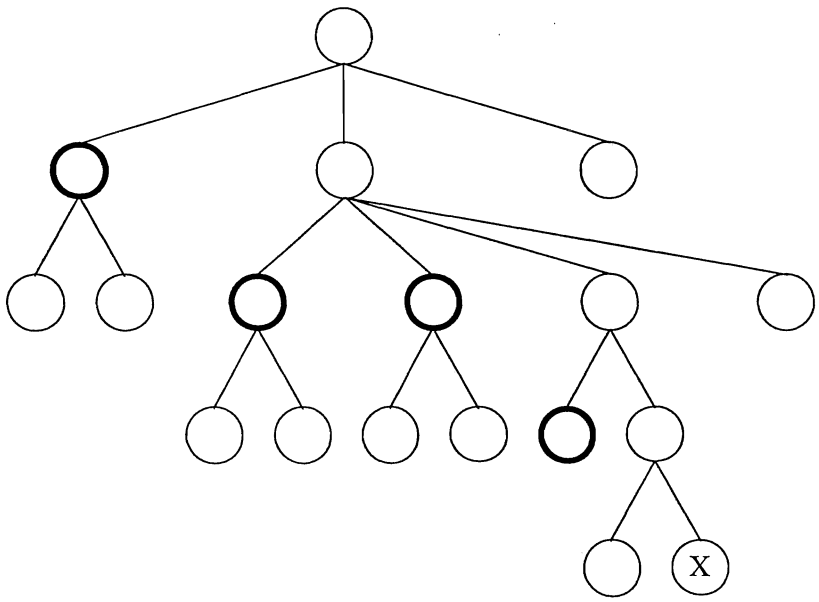

Figure 6. A conceptual search tree

Consequently, all the information which is needed to test this dominance rule can be saved very efficiently as follows: When the procedure backtracks into a node at level p of the search tree, save the set of added precedence relations of that node (with respect to the original problem, not with respect to the parent node) on a stack. Before doing that, however, delete from the stack all the information which has been saved on level $p+1$. This guarantees that all the required information for dominance testing is saved on the stack, and removed when it is no longer required. The set of added precedence constraints from nodes that are feasible (yielding an upper bound), time-infeasible (because maximal time lags are violated) or fathomed need not be saved on the stack since all other nodes which contain that set as a subset will also be feasible, time-infeasible or fathomed.

4.2.5. Reducing the solution space using preprocessing

Before initiating the branch-and-bound procedure, the solution space can be reduced by simultaneously examining the GPRs and the resource requirements. If on the one hand, two activities i and j can never be companions due to the resource constraints, but on the other hand, the (generalized) precedence relations allow for an overlap, then the precedence relations can sometimes be tightened in order to avoid the overlap. For instance, suppose that in the problem example of Fig. 2, activities 2 and 3 cannot be companions due to their resource requirements. However, the maximal distances between activities 2 and 3 are $d_{2,3} = -\infty$ and $d_{3,2} = -1$, indicating that an overlap is allowed. The only restriction is a maximal time lag between activity 2 and 3 of 1 time unit, implying that activity 2 can only start 1 time unit before activity 3. However, because activities 2 and 3 are no companions, they can never be scheduled in parallel (notice that activity 2 cannot start 1 time unit before activity 3 because then an overlap would occur), such that the time lag between activities 3 and 2 can be increased to $l_{3,2} = d_3 = 2$.

THEOREM 6. If $\exists\, i, j \in V$ *and resource type* k *for which* $r_{ik} + r_{jk} > a_k$ *and* $-d_j < d_{ij} < d_i$, *we can set* $l_{ij} = d_i$ *without changing the optimal solution of the RCPSP-GPR.*

PROOF. See appendix.

This rather straightforward rule is often very effective in reducing the solution space. Moreover, because it can be executed as a preprocessing rule, it is very efficient.

4.3. The branch-and-bound algorithm

The detailed algorithmic steps of the proposed branch-and-bound algorithm are described below. The maximal distance between two activities i and j is given by $d[p][i][j]$, where p denotes the level in the search tree. For each such level, a distance matrix $d[p]$ will have to be stored. For ease of reference, a numerical example is given in Appendix 1.

STEP 1: INITIALISATION

Let $T = 9999$ be an upper bound on the project duration.

Set the level of the branch-and-bound tree $p = 0$.

Compute the constraint digraph *cd* (using the transformation rules discussed in section 2).

Compute $d[0]$, the distance matrix at level 0 using the Floyd-Warshall algorithm ($O[n^3]$).

If the project is not time-feasible (i.e. $\exists\, i \in V: d[0][i][i] > 0$), STOP.

Preprocessing: reduce the solution space by adjusting $d[0]$:

$\forall (i,j) \mid i, j \in V$ and $\exists$ resource type $k: r_{ik} + r_{jk} > a_k$ and

case 1: $-d_j < d[0][i][j] < d_i$, set $l_{ij} = d_i$

case 2: $-d_i < d[0][j][i] < d_j$, set $l_{ji} = d_j$

Recompute $d[0]$ using the Floyd-Warshall algorithm ($O[n^3]$).

Compute the critical path based lower bound $lb_0 = d[0][1][n]$ and go to STEP 3.

STEP 2: TEMPORAL ANALYSIS

Compute $d[p]$, the extended distance matrix at level p as follows:

$\forall i, j \in V:\ d[p][i][j] = d[p-1][i][j]$. $\forall i, j \in V,\ l \in D_d:\ d[p][i][j] =$

$\max\{d[p][i][j], d[p-1][i][k] + d_k + d[p-1][l][j]\}$, k being the delaying activity ($O[n^2]$).

If $T < 9999$, compute lb_3^g (using the algorithm described in section 4.2.).

If $lb_3^g \geq T$, erase the delaying mode and go to STEP 6.

STEP 3: RESOURCE ANALYSIS

Determine the *first* period in which a resource conflict occurs, i.e. the first period $]t^*\text{-}1, t^*]$ for which $\sum_{i \in S(t^*)} r_{ik} > a_k$ for some resource type k. $S(t^*)$, the set of activities in progress in period $]t^*\text{-}1, t^*]$, is called the *conflict set*.

If there is no conflict, let $T = \min\{T, d[p][1][n]\}$, erase the delaying mode and go to STEP 6.

Store the distance matrix.

STEP 4: DETERMINE MINIMAL DELAYING ALTERNATIVES AND MINIMAL DELAYING MODES

Increase the branch level of the search tree: $p = p + 1$.

Determine the minimal delaying set, i.e. the set of minimal delaying alternatives:

$$D = \left\{ D_d \,\middle|\, D_d \subset S(t^*) \text{ and } \forall \text{ resource type } k : \sum_{i \in S(t^*)} r_{ik} - \sum_{i \in D_d} r_{ik} \le a_k \text{ and } \forall\, D_{d'} \in \mathrm{D} : D_{d'} \not\subset D_d \right\}$$

Extend all minimal delaying alternatives using Theorem 2 and eliminate all non-minimal delaying alternatives. Determine the set of minimal delaying modes:

$$M = \left\{ M_m \,\middle|\, M_m = \left\{ k \prec D_d \right\}, k \in S(t^*) \setminus D_d, D_d \in D \right\}.$$

Eliminate all delaying modes satisfying Theorem 3.

Arbitrarily select a delaying mode M_m with corresponding delaying alternative D_d.

STEP 5: EVALUATE DELAYING MODES

For all delaying modes M_m {

If the precedence constraints cannot be added, i.e. $\exists\, l \in D_d$: $k \prec l$ is infeasible, i.e.

$d_k > -d[p][l][k]$ (k being the delaying activity), continue with next delaying mode M_m.

Compute lb_0 as follows: Set $lb_0 = d[p-1][1][n]$. $\forall\, l \in D_d$ and delaying activity k:

$$lb_0 = \max\left\{ lb_0, d[p-1][1][k] + d_k + d[p-1][l][n] \,\middle|\, j \in D_d \right\}.$$

If $lb_0 \ge T$, continue with next delaying mode M_m.

If the set of added precedence constraints of a previously examined node saved earlier is a subset of the set of added precedence constraints of the current node, continue with next delaying mode M_m.

Calculate $lb = lb_0$.

Temporarily store the delaying mode and its lower bound lb.

}

STEP 6: BRANCHING

If no delaying modes are left to branch from at level p, go to STEP 7.

Select the delaying mode M_m with the smallest lower bound $lb = lb_0$ (arbitrary tie-break).

If $lb \ge T$, erase all remaining delaying modes at level p and go to STEP 7.

Go to STEP 2.

STEP 7: BACKTRACKING

Decrease the branch level of the search tree: $p = p - 1$.

If $p \le 0$, STOP with the optimal solution with a makespan of T

(if T = 9999, then there exists no feasible solution).

Delete from the stack the information which has been previously saved on level p+1 for dominance testing.

Save the necessary information for node dominance testing on the stack, i.e. the list of added precedence constraints of the node reached upon backtracking.

Erase the distance matrix and the lower bound of this node and go to STEP 6.

4.4. A numerical example

Consider the example given in Fig. 2. Suppose there are two renewable resource types with a constant availability of 10 units each, and resource requirements for each of the activities equal to $\{0, 6, 5, 5, 6, 5, 3, 3, 2, 5, 0\}$ and $\{0, 1, 3, 2, 7, 6, 3, 1, 1, 2, 0\}$ respectively. We will compute the optimal solution by going through the steps of the algorithm.

STEP 1: T = 9999 and p = 0. Compute $d[0]$. $lb_0 = d[0][1][11] = 12$. The project is time-feasible.
Reduce solution space, i.e. set: $l_{3,2} = d_3 = 2$ ($-d_2(=-3) < d[0][3][2] (=-1) < d_3(=2)$),
$l_{4,2} = d_4 = 6$ ($-d_2(=-3) < d[0][4][2] (=2) < d_4(=6)$),
$l_{6,2} = d_6 = 3$ ($-d_2(=-3) < d[0][6][2] (=-1) < d_6(=3)$), and
$l_{5,10} = d_5 = 4$ ($-d_{10}(=-5) < d[0][5][10] (=3) < d_5(=4)$).
Recompute $d[0]$, $lb_0 = d[0][1][11] = 16$.

STEP 3: There is a resource conflict in period]0,1]; $S(1) = \{3,4,6\}$. Store the distance matrix.

STEP 4: p = 1. $D = \{\{3\},\{4\},\{6\}\}$. Because $d[0][3][6] \geq 0$, Theorem 2 can be applied: extend delaying alternative $\{3\}$ with activity 6: $D = \{\{3,6\},\{4\},\{6\}\}$. Eliminate non-minimal delaying alternative $\{3,6\}$: $D = \{\{4\},\{6\}\}$; $M = \{\{3 \prec 4\},\{6 \prec 4\},\{3 \prec 6\},\{4 \prec 6\}\}$. Because $d[0][3][6] + d_6(=0+3) > d_3(=2)$, we can eliminate mode $\{6 \prec 4\}$ since it is dominated by delaying mode $\{3 \prec 4\}$: $M = \{\{3 \prec 4\},\{3 \prec 6\},\{4 \prec 6\}\}$.

STEP 5: Mode $\{3 \prec 4\}$: The precedence constraint $3 \prec 4$ can be added. Initialize
$lb_0 = d[0][1][11] = 16$. $lb_0 = \max\{lb_0, d[0][1][3] + d_3 + d[0][4][11]\} = \max\{16, 0+2+16\} = 18$.
$lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 18$. Mode $\{3 \prec 6\}$: The precedence constraint $3 \prec 6$ can be added. Initialize
$lb_0 = d[0][1][11] = 16$. $lb_0 = \max\{lb_0, d[0][1][3] + d_3 + d[0][6][11]\} = \max\{16, 0+2+13\} = 16$.
$lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 16$. Mode $\{4 \prec 6\}$: The precedence constraint $4 \prec 6$ can be added. Initialize
$lb_0 = d[0][1][11] = 16$. $lb_0 = \max\{lb_0, d[0][1][4] + d_4 + d[0][6][11]\} = \max\{16, 0+6+13\} = 19$.
$lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 19$.

STEP 6: Select the delaying mode with the smallest lb, i.e. $\{3 \prec 6\}$. $lb < T$.

STEP 2: Compute $d[1]$. T is still equal to 9999.

STEP 3: There is a resource conflict in period]2,3]: $S(3) = \{4,5,6\}$. Store the distance matrix.

STEP 4: $p = 2$. $D = \{\{5\},\{4,6\}\}$. $M = \{\{4 \prec 5\},\{6 \prec 5\},\{5 \prec 4,\ 5 \prec 6\}\}$.

STEP 5: Mode $\{4 \prec 5\}$: The precedence constraint $4 \prec 5$ can be added. Initialize $lb_0 = d[1][1][11] = 16$. $lb_0 = \max\{lb_0, d[1][1][4] + d_4 + d[1][5][11]\} = \max\{16, 0+6+9\} = 16$. Note that $d[1][5][11] = 9$ instead of 8 because of the preprocessing performed in STEP 1. $lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 16$. Mode $\{6 \prec 5\}$: The precedence constraint $6 \prec 5$ can be added. Initialize $lb_0 = d[1][1][11] = 16$. $lb_0 = \max\{lb_0, d[1][1][6] + d_6 + d[1][5][11]\} = \max\{16, 2+3+9\} = 16$. $lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 16$. Mode $\{5 \prec 4,\ 5 \prec 6\}$: The precedence constraints can be added. Initialize $lb_0 = d[1][1][11] = 16$. $lb_0 = \max\{lb_0, d[1][1][5] + d_5 + d[1][4][11]\} = \max\{16, 2+4+16\} = 22$. $lb_0 = \max\{lb_0, d[1][1][5] + d_5 + d[1][6][11]\} = \max\{22, 2+4+13\} = 22$. $lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 22$.

STEP 6: Select the delaying mode with the smallest lb, i.e. $\{4 \prec 5\}$ (arbitrary tie-break with $\{6 \prec 5\}$). $lb < T$.

STEP 2: Compute $d[2]$. T is still equal to 9999.

STEP 3: There is a resource conflict in period]6,7]: $S(7) = \{2,5,9\}$. Store the distance matrix.

STEP 4: $p = 3$. $D = \{\{2\},\{5\}\}$. $M = \{\{5 \prec 2\},\{9 \prec 2\},\{2 \prec 5\},\{9 \prec 5\}\}$.

STEP 5: Mode $\{5 \prec 2\}$: The precedence constraint $5 \prec 2$ can be added. Initialize $lb_0 = d[2][1][11] = 16$. $lb_0 = \max\{lb_0, d[2][1][5] + d_5 + d[2][2][11]\} = \max\{16, 6+4+10\} = 20$. $lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 20$. Mode $\{9 \prec 2\}$: The precedence constraint $9 \prec 2$ can be added. Initialize $lb_0 = d[2][1][11] = 16$. $lb_0 = \max\{lb_0, d[2][1][9] + d_9 + d[2][2][11]\} = \max\{16, 6+5+10\} = 21$. $lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 21$. Mode $\{2 \prec 5\}$: The precedence constraint $2 \prec 5$ cannot be added since $d_2 (= 3) > -d[2][5][2] (= 2)$. Mode $\{9 \prec 5\}$: The precedence constraint $9 \prec 5$ can be added. Initialize $lb_0 = d[2][1][11] = 16$. $lb_0 = \max\{lb_0, d[2][1][9] + d_9 + d[2][5][11]\} = \max\{16, 6+5+9\} = 20$. $lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 20$.

STEP 6: Select the delaying mode with the smallest lb, i.e. $\{9 \prec 5\}$ (arbitrary tie-break with $\{5 \prec 2\}$). $lb < T$.

STEP 2: Compute $d[3]$. T is still equal to 9999.

STEP 3: There is a resource conflict in period]11,12]: $S(12) = \{2,5\}$. Store the distance matrix.

STEP 4: $p = 4$. $D = \{\{2\},\{5\}\}$. $M = \{\{5 \prec 2\},\{2 \prec 5\}\}$.

STEP 5: Mode $\{5 \prec 2\}$: The precedence constraint $5 \prec 2$ can be added. Initialize $lb_0 = d[3][1][11] = 20$. $lb_0 = \max\{lb_0, d[3][1][5] + d_5 + d[3][2][11]\} = \max\{16, 11+4+10\} = 25$. $lb_0 < T$. No dominance information has been saved yet. Store the delaying mode and its lower bound $lb = lb_0 = 25$. Mode $\{2 \prec 5\}$: The precedence constraint $2 \prec 5$ cannot be added since $d_2 (= 3) > -d[3][5][2] (= 2)$. This node is also dominated by node $\{2 \prec 5\}$ on level 3. However, it is not necessary to save the dominance information from that node because any node

dominated by a time-infeasible node will also be time-infeasible. The same applies to (time- and resource-) feasible nodes.

STEP 6: Select the only remaining delaying mode $\{5 \prec 2\}$. $lb < T$.

STEP 2: Compute $d[4]$. T is still equal to 9999.

STEP 3: No resource conflict. $T = \min\{T, d[4][1][n]\} = \min\{9999, 25\} = 25$. Erase the mode.

STEP 6: No delaying modes are left to branch from at level 4.

STEP 7: Decrease the branch level of the search tree: $p = 3$ (> 0). Save the necessary information for node dominance testing on the stack, i.e. the list of added precedence constraints of the node $\{9 \prec 5\}$ on level 3. Erase the distance matrix of that node and its lower bound lb.

STEP 6: Select the delaying mode with the smallest lower bound lb, i.e. $\{5 \prec 2\}$. $lb < T$.

STEP 2: Compute $d[3]$. $lb_3^g = 20 < T(= 25)$. Notice that the computation of lb_3^g has been deferred till this point in the search process.

STEP 3: No resource conflict. Set $T = \min\{T, d[3][1][n]\} = \min\{9999, 20\} = 20$. Erase the mode.

STEP 6: Select the only remaining delaying mode $\{9 \prec 2\}$. $lb(= 21) \geq T(= 20)$. Therefore, erase all remaining delaying modes at level 3 (which is only the current delaying mode).

STEP 7: Decrease the branch level of the search tree: $p = 2$ (> 0). Delete from the stack the information which has previously been saved on level 3 for dominance testing, i.e. the set of added precedence constraints of node $\{9 \prec 5\}$. Save the necessary information for node dominance testing on the stack, i.e. the list of added precedence constraints of the node $\{4 \prec 5\}$ on level 2. Erase the distance matrix of that node and its lower bound lb.

STEP 6: Select the delaying mode with the smallest lb, i.e. $\{6 \prec 5\}$. $lb < T$.

STEP 2: Compute $d[2]$. $lb_3^g = 18 < T(= 20)$.

STEP 3: There is a resource conflict in period]5,6]: $S(6) = \{4,5\}$. Store the distance matrix.

STEP 4: $p = 3$. $D = \{\{4\},\{5\}\}$. $M = \{\{5 \prec 4\},\{4 \prec 5\}\}$.

STEP 5: Mode $\{5 \prec 4\}$: The precedence constraint $5 \prec 4$ can be added. Initialize $lb_0 = d[2][1][11] = 16$. $lb_0 = \max\{lb_0, d[3][1][5] + d_5 + d[3][4][11]\} = \max\{16, 5 + 4 + 16\} = 25$. $lb_0 < T$. Mode $\{4 \prec 5\}$: The precedence constraint $4 \prec 5$ can be added. Initialize $lb_0 = d[2][1][11] = 16$. $lb_0 = \max\{lb_0, d[3][1][4] + d_4 + d[3][5][11]\} = \max\{16, 0 + 6 + 9\} = 16$. $lb_0 < T$. This node is dominated by node $\{4 \prec 5\}$ on level 2 and can therefore be fathomed.

STEP 6: No delaying modes are left to branch from at level 3.

STEP 7: Decrease the branch level of the search tree: $p = 2$ (> 0). Save the necessary information for node dominance testing on the stack, i.e. the list of added precedence constraints of the node $\{6 \prec 5\}$ on level 2. Erase the distance matrix of that node and its lower bound lb.

STEP 6: Select the delaying mode with the smallest lb, i.e. $\{5 \prec 4, 5 \prec 6\}$. $lb(= 22) \geq T(= 20)$. Therefore, erase all remaining delaying modes at level 3 (only the current delaying mode).

STEP 7: Decrease the branch level of the search tree: $p = 1$ (> 0). Delete from the stack the information which has previously been saved on level 2 for dominance testing, i.e. the set of added precedence constraints of nodes $\{4 \prec 5\}$ and $\{6 \prec 5\}$. Save the necessary information for node dominance testing on the stack, i.e. the list of added precedence constraints of the node $\{3 \prec 6\}$ on level 2. Erase the distance matrix of that node and its lower bound lb.

STEP 6: Select the delaying mode with the smallest lb, i.e. $\{3 \prec 4\}$. $lb < T$.
STEP 2: Compute $d[1]$. $lb_3^g = 20 \geq T(= 20)$.
STEP 6: Select the only remaining delaying mode $\{4 \prec 6\}$. $lb < T$.
STEP 2: Compute $d[1]$. $lb_3^g = 21 \geq T(= 20)$.
STEP 6: No delaying modes are left to branch from.
STEP 7: Decrease the branch level of the search tree: p = 0. Stop with the optimal solution with a makespan of 20.

The search tree of the example is given in Fig. 7. The nodes contain several labels, indicating the node number, the added precedence constraints, the applicable lower bounds and the obtained upper bound (for the nodes representing a feasible solution). All fathomed nodes are indicated by dashed lines. The reason for fathoming is indicated below the nodes: P (*P*recedence) means the node represents a time-infeasible network; lb_0 means that $lb_0 \geq T$; lb_3 means that $lb_3^g \geq T$ and D means that the node was dominated. $T = x$ means that a feasible solution with a makespan of x was found.

The number of nodes in the search tree (including dominated nodes, but apart from the root node) is equal to 14. The nodes actually branched from equals 4. Table II shows the impact of the node fathoming rules on the number of nodes created and branched from for the example. Notice, however, that the node fathoming rules in Table II are entered into the procedure in a sequential manner. Therefore, Table II does not reveal the power of each of the node fathoming rules. Estimating the impact of each of the node fathoming rules requires a full factorial experiment, in which all the combinations of the node fathoming rules are examined. This will be discussed in section 5. The resource profile of the optimal solution with respect to the first resource type is given in Fig. 8.

Table II. The impact of the node fathoming rules

Node fathoming rule	Created nodes	Branched nodes
complete enumeration	1709	325
minimal delaying alternatives (Theorem 1)	375	104
lb_0	141	41
redundant delaying alternatives (Theorem 2)	73	33
redundant delaying modes (Theorem 3)	62	28
lb_3^g (Theorem 4)	25	10
subset dominance rule (Theorem 5)	25	10
solution space reduction (Theorem 6)	14	4

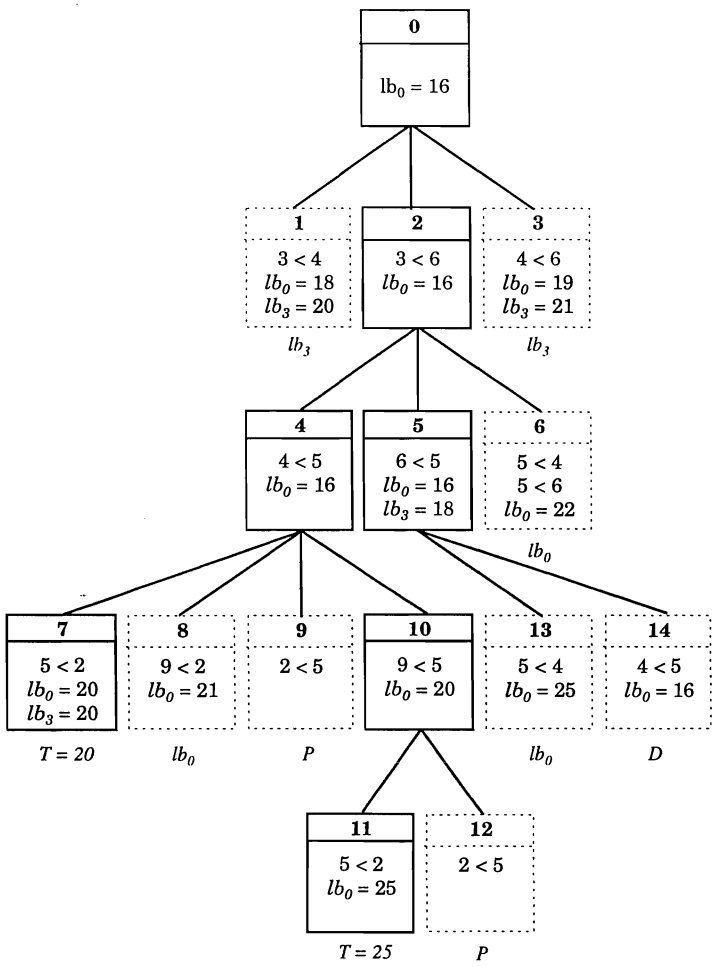

Figure 7. The search tree for the example

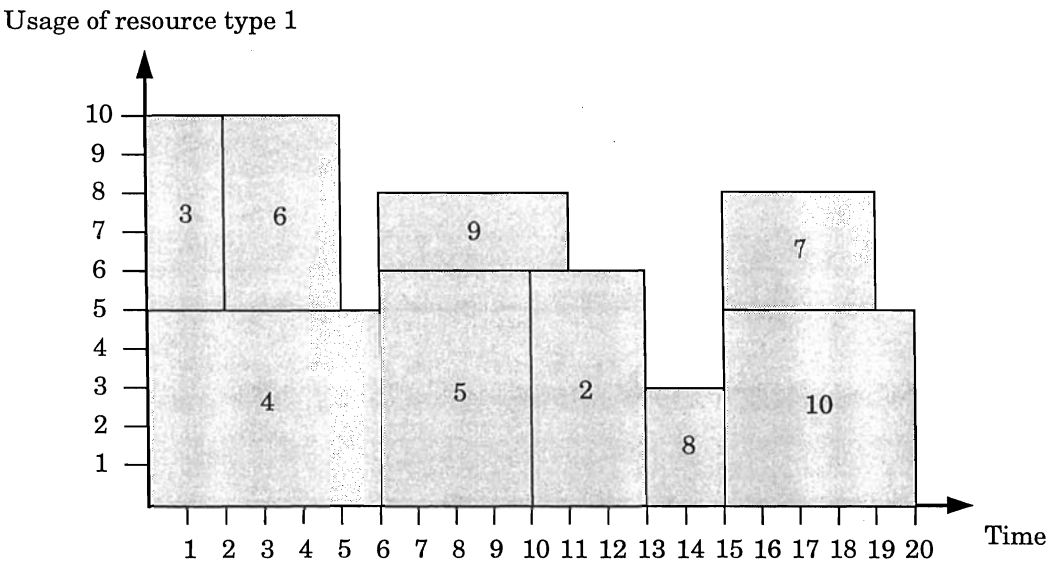

Figure 8. The optimal resource profile of resource type 1

4.5. Another example

The following example (Demeulemeester, 1992) is a multi-project scheduling problem, in which three separate projects, each with a release date and a deadline, have to be scheduled, while subject to time-varying resource constraints. Fig. 9 clearly displays the three projects, which are connected to the dummy start and end activities such that one single project is obtained. The numbers above the nodes indicate the activity durations, the numbers below each node indicate the resource requirements for two renewable resources. The project is given in standardized form, so that the time lag values are all minimal start-start precedence relations. Notice that the time lags between activity 1 and activities 2, 7 and 11 (start-start lag of zero), and between activities 6, 10 and 13 and activity 14 (start-start lag equal to the duration of the predecessor activity) are needed to ensure that the (dummy) start and finish activities 1 and 14 correspond with the start and completion of the project. The release dates and deadlines are given in Table III. The resource availabilities vary over time as indicated in Table IV.

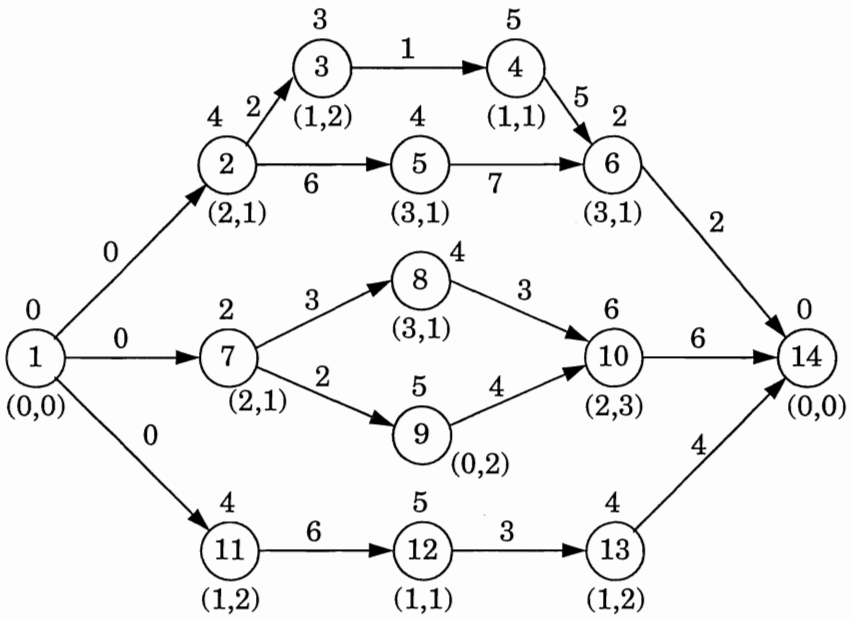

Figure 9. A multi-project GRCPSP

Table III. Release dates and deadlines for the three projects

Project	Activities	Release date	Deadline
1	2, 3, 4, 5, 6	5	20
2	7, 8, 9, 10	0	15
3	11, 12, 13	0	25

Table IV. Resource availabilities

Time interval	Availability resource type 1	Availability resource type 2
0 - 7	6	5
7 - 12	7	5
12 - 18	7	6
18 - end	5	5

Notice that the problem is a generalized resource-constrained project scheduling problem (GRCPSP, Demeulemeester and Herroelen, 1996a). This implies that all time lags are minimal time lags only (with the additional assumption that a successor can never start before its predecessor), and that release dates, due dates and time-varying resource availabilities are allowed. Computational experience with the optimal procedure of Demeulemeester and Herroelen (1996a) is given in Demeulemeester (1992). Because the RCPSP-GPR is more general than the GRCPSP, the latter can also be solved using our solution procedure for the RCPSP-GPR. In the RCPSP-GPR, however, release dates, due dates and time-varying resource availabilities need not be specified separately. Release dates and due dates can be modeled by appropriate minimal and maximal time lags, and time-varying resource availabilities can be modeled by dummy activities with fixed starting times which absorb a certain amount of each renewable resource. The resulting problem is given in Fig. 10.

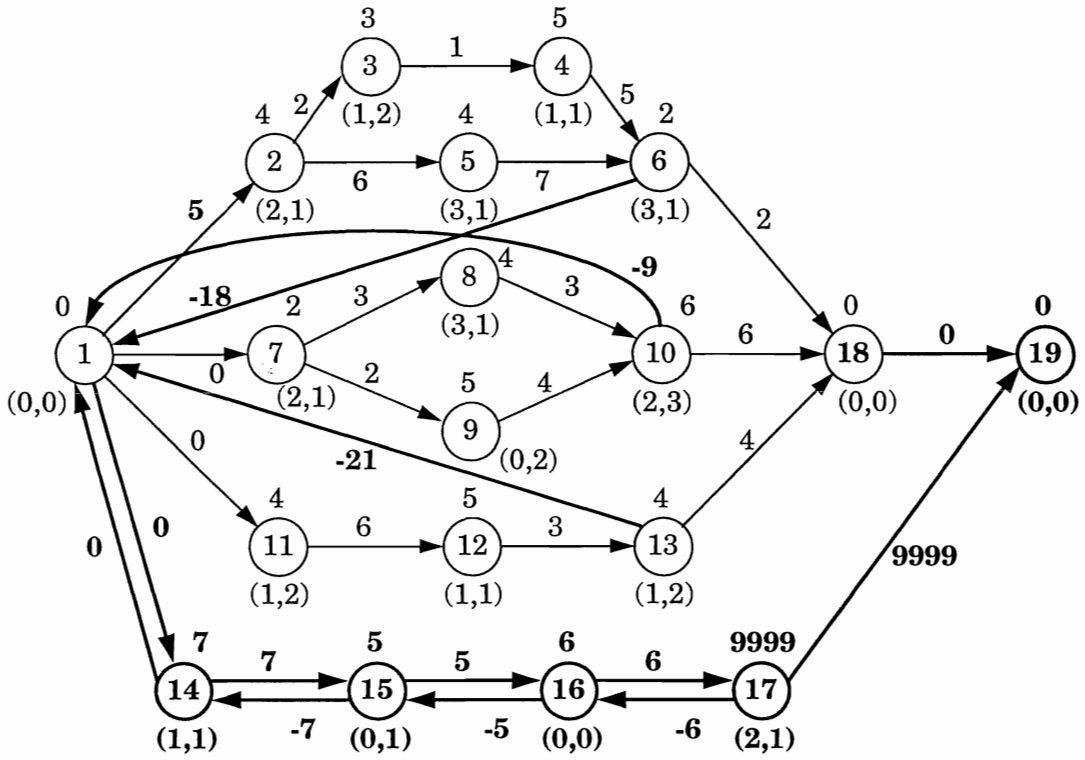

Figure 10. The corresponding RCPSP-GPR

Notice that dummy activity 17, which is to absorb a certain amount of both resource types up to the project completion, has been given an infinite duration (represented by a high number 9999), since the project length is not yet known. This complicates the problem somewhat since the objective is now, not to minimize the project makespan (given by the completion time of dummy activity 19), but to minimize the completion time of dummy activity 18 (which represents the maximum of the completion times of each of the three individual projects). Since the procedure can be readily adapted for any regular measure of performance (see section 4.6.1), and minimizing the completion time of an activity (or a weighted function of the completion times of several activities) is a regular objective function, this poses no real problem. The changes to the original problem are indicated in bold.

The search tree obtained with our procedure is given in Fig. 11. Apart from the root node, 11 nodes were created in the tree, while 3 nodes are actually branched from. As a comparison, using the procedure of Demeulemeester and Herroelen (1996a), 14 nodes are created, while 4 nodes are actually branched from. The resource profiles of the optimal solution with respect to both resource types are given in Fig. 12 and 13. Notice that, at the first level of the search tree, the number of delaying modes is quite low, whereas the number of activities in the conflict set is quite high: $S(7) = \{2,8,9,10,12,14\}$. The reason can be found in the application of Theorems 2 and 3. The delaying set is : $D = \{\{2,8,12\}, \{2,8,14\}, \{2,9\}, \{2,12,14\}, \{8,9\}, \{8,12,14\}, \{9,12,14\}, \{10\}\}$. This would, normally, give rise to 28 minimal delaying modes. However, application of Theorem 2 reveals that many of the minimal delaying alternatives can be extended as follows:
$D' = \{\{2,8,10,12\}, \{2,8,9,10,12,14\}, \{2,9,10\}, \{2,8,9,10,12,14\}, \{8,9,10\}, \{2,8,9,10,12,14\}, \{2,8,9,10,12,14\}, \{10\}\}$
Therefore, when we eliminate all non-minimal delaying alternatives, we are only left with one minimal delaying alternative: $D = \{\{10\}\}$, which results in five minimal delaying modes:
$M = \{\{2 \prec 10\}, \{8 \prec 10\}, \{9 \prec 10\}, \{12 \prec 10\}, \{14 \prec 10\}\}$. Furthermore, using Theorem 3, we can eliminate delaying modes $\{2 \prec 10\}$ and $\{12 \prec 10\}$ because $d_{14,2} + d_2 = 5 + 4 > d_{14} = 7$ and $d_{14,12} + d_{12} = 6 + 5 > d_{14} = 7$. Consequently, the set of minimal delaying modes equals
$M = \{\{8 \prec 10\}, \{9 \prec 10\}, \{14 \prec 10\}\}$.

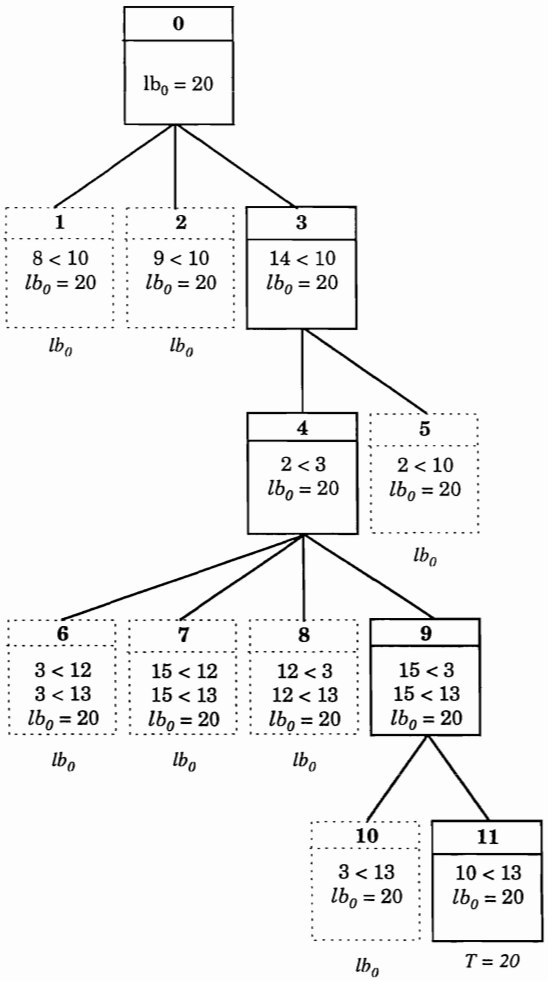

Figure 11. The search tree for the multiproject example

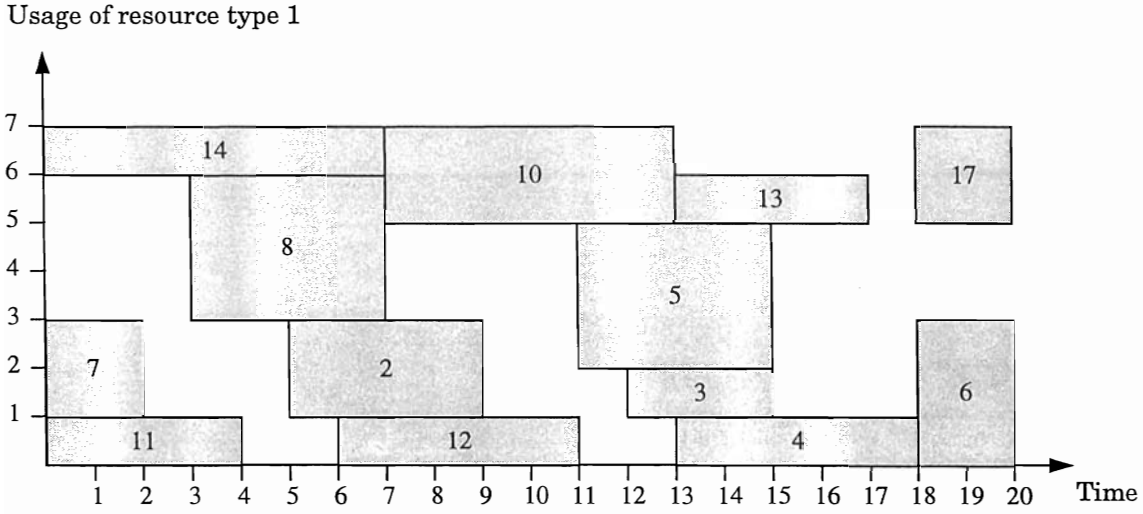

Figure 12. The optimal resource profile of resource type 1

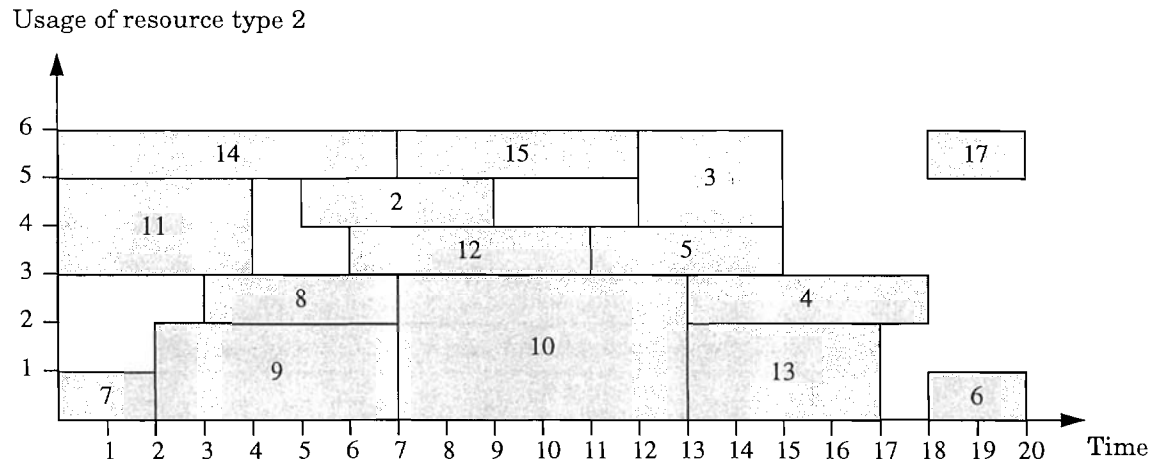

Figure 13. The optimal resource profile of resource type 2

4.6. Other measures of performance

4.6.1. Regular measures of performance

Two slight modifications are needed to extend the procedure to other regular measures of performance, i.e. a non-decreasing function of the activity completion times. For regular measures of performance the evaluation of nodes in the search tree can be based on early start schedules. So first, we need to replace lb_0 by the new measure. Second, lb_3^g can no longer be used as a node fathoming rule.

4.6.2. Nonregular measures of performance

Even non-regular measures of performance, such as the maximization of the net present value (*npv*) of the project, can be handled. However, the project network in each node cannot be evaluated using its early start schedule. When maximizing the *npv* of the project, in which cash flows are associated with each activity, the evaluation of each node can be accomplished using a payment scheduling problem (Russell 1970), i.e. maximizing the net present value of the corresponding (time-feasible, but not necessarily resource-feasible) project without taking the resource constraints into account. Algorithms for the payment scheduling problem can be found in Russell (1970, Grinold (1972), Elmaghraby and Herroelen (1990), Herroelen and Gallens (1993) and Herroelen et al. (1996). Naturally, these algorithms need to be extended to cope with generalized precedence relations. Also, lb_3^g cannot be used as a node fathoming rule. Other non-regular measures of performance, such as resource levelling objective functions, can be handled as long as the networks (without resource constraints) in each node can be evaluated and optimized.

5. Computational experience

5.1. Benchmark problem set

The procedure has been programmed in Microsoft® Visual C++ 2.0 under Windows NT for use on a Digital Venturis Pentium-60 personal computer. In order to validate our branch-and-bound procedure, we generated a number of RCPSP-GPR instances based on the problem set for the RCPSP assembled by Patterson (1984). This problem set consists of 110 RCPSP instances with a number of activities ranging from 7 to 51. We selected only the 55 smaller problem instances with less than 25 activities and extended each of them to incorporate GPRs. Since the hardness of the RCPSP-GPR will undoubtedly vary with the (relative) number of maximal time lags and the nature of the GPRs, we generated 10 sets of 55 problem instances based on different values for the percentage of maximal time lags (*A*), the percentage of precedence relations that allow for activity overlaps (*B*) and the tightness of the maximal time lags (*C*). Table V shows how the 550 problems are derived.

Table V. The RCPSP-GPR benchmark problem set

A	0%		10%				20%			
B	0%	100%	0%		100%		0%		100%	
C			loose	tight	loose	tight	loose	tight	loose	tight
CLASS	1*	2**	3	4	5	6	7	8	9	10

* RCPSP instances
** GRCPSP instances

Since the problem set of Patterson (1984) was originally developed as a RCPSP benchmark set, all the precedence relations are of the zero-lag finish-start type. Using a similar approach as Demeulemeester and Herroelen (1996a) for the GRCPSP, we transform for a number of activities i and j the zero-lag finish-start precedence relations to start-start precedence relations with a time lag chosen randomly between 0 and $2d_i$ (such that the expected value still equals d_i, which corresponds to a zero-lag finish-start precedence relation).

Maximal precedence relations are added to each problem instance as follows: Two activities i and j are selected randomly. If a maximal time lag can be added between i and j, i.e. if no cycles of positive length (which would result in a time-infeasibility) would be created, a maximal time lag is added with a value chosen randomly between 1 and the makespan of the project without the resource constraints. To get tighter maximal time lags, however, a similar procedure is followed but with a maximal value equal to half the project makespan. This

procedure is repeated until the required percentage of maximal time lags is reached. Notice that infeasible problems may be generated, since checking whether a problem is feasible or not is NP-complete.

5.2. Computational results

5.2.1. Impact of node fathoming: a comparative experiment

Table VI shows how the node fathoming rules affect the average required computation time (in seconds) and the average number of nodes created and branched from in the search tree when solving all 550 problems to optimality. The node fathoming rules are entered sequentially. Here and in the subsequent tables, the numbers between brackets denote the standard deviations. A time limit of 1 hour was imposed for the complete enumeration algorithm and the algorithm based only on minimal delaying alternatives, since many problems could not be solved within a reasonable amount of time. Therefore, in rows 1 and 2, the average computation time and (especially) the standard deviation are heavily deflated. For the complete enumeration algorithm, 324 problems could not be solved within 1 hour, whereas when using Theorem 1,251 problems could not be solved within the time limit.

Table VI. The impact of the node fathoming rules

Node fathoming rule	Computation time (seconds)	Created nodes	Nodes branched from
complete enumeration	2,188.54	4,658,047	775,455
	(1,726.26)	(6,612,913)	(864,721)
minimal delaying alternatives (Theorem 1)	1,766.16	3,267,513	702,835
	(1,733.58)	(3,881,148)	(786,870)
lb_0	276.27	1,236,804	265,154
	(3,834.47)	(16,009,078)	(3,694,834)
redundant delaying alternatives (Theorem 2)	90.50	348,801	89,614
	(834.79)	(2,793,111)	(776,252)
redundant delaying modes (Theorem 3)	59.25	237,270	57,150
	(455.49)	(1,738,012)	(372,317)
lb_3^g (Theorem 4)	39.65	134,946	28,381
	(365.27)	(1,064,435)	(199,179)
subset dominance rule (Theorem 5)	18.37	60,909	13,483
	(134.27)	(373,094)	(81,392)
solution space reduction (Theorem 6)	10.12	39,916	8091
	(42.67)	(131,348)	(32,199)

Table VII : Detailed performance analysis on the 10 problem classes

% maximal time lags	0%		10%				20%			
% minimal time lags	0%	100%	0%		100%		0%		100%	
max. time lags values			loose	tight	loose	tight	loose	tight	loose	tight
CPU-time (seconds)	24.72	26.61	10.96	9.63	9.01	9.26	5.41	1.70	3.24	0.64
	(56.44)	(103.20)	(30.09)	(31.99)	(22.72)	(29.75)	(16.45)	(5.42)	(17.45)	(2.50)
created nodes	110,525	77,566	49,493	35,847	37,933	37,026	26,875	6594	15,628	1,675
	(235,006)	(208,228)	(138,546)	(102,154)	(93,659)	(114,900)	(82,017)	(21,389)	(89,147)	(6,065)
branched nodes	18,128	21,234	8,598	6,908	8,507	8,625	4,295	1,502	3,170	640
	(40,735)	(75,070)	(25,114)	(21,642)	(19,976)	(28,216)	(14,367)	(4,972)	(17,206)	(2,521)

5.2.2. Effect of problem characteristics

Table VII gives a more detailed view on the class-by-class performance of our procedure with all node fathoming rules included on the 550 problem instances. Clearly, the percentage of maximal precedence relations, their tightness and the percentage of precedence relations that allow for activity overlaps have a significant impact on the computational effort. From Table VII it is clear that the higher the number of maximal time lags and the higher the number of minimal time lags that allow for activity overlaps, the more efficient our procedure becomes. This result is very logical since for the problem class with no maximal time lags and no minimal time lags that allow for activity overlaps (class 1), many of the studied node fathoming rules become useless. Moreover, since the problems of class 1 are actually RCPSP instances, dedicated optimal procedures, such as the procedure of Demeulemeester and Herroelen (1992, 1995) can be used. Notice that the problems of class 2 are GRCPSP instances, for which the procedure of Demeulemeester and Herroelen (1996a) can be used.

Notice that the impact of the number of minimal time lags that allow for activity overlaps on the required computational effort differs from class to class. If we compare class 1 (RCPSP instances) to class 2 (GRCPSP), we can see that, in accordance to the results of Demeulemeester and Herroelen (1996a), the computational effort needed to solve the GRCPSP, using their optimal solution procedure for the GRCPSP, increases when the relative number of overlapping precedence relations increases. However, if we compare classes 3 and 5, 4 and 6, 7 and 9 or 8 and 10, we can see that the inclusion of minimal time lags that allow for activity overlaps reduces the computational effort. This decrease in required CPU time (and nodes) is largest for problem class 10 (relative to class 8), i.e. when many (tight) maximal time lags are also included.

5.2.3. Impact of node fathoming as a function of problem characteristics

We can see from Table VI that the inclusion of node fathoming rules siginificantly reduces the computational effort to solve all 550 RCPSP-GPR instances. However, Table VII clearly shows that the performance of our procedure heavily depends on the characteristics of the problem instances. The higher the number of maximal time lags and the higher the number of minimal time lags that allow for activity overlaps, the more efficient our procedure. Therefore, a more detailed analysis is required to reveal how the impact of the inclusion of node fathoming rules depends on the percentage and characteristics of the maximal time lags and the nature of the minimal time lags. As an example, Table VIII shows how the inclusion of the node fathoming rules affects the computational effort for the problems of class 10. When we compare Table VIII to Table VI, it is clear that the node fathoming rules have a much higher impact on the computational effort needed to solve the problem instances of class 10 than for all 550 problem

instances. For the complete enumeration algorithm, 27 problems could not be solved within 1 hour, whereas when using Theorem 1, 14 problems could not be solved within the time limit.

Table VIII. The impact of the node fathoming rules for problem class 10

Node fathoming rule	Computation time (seconds)	Created nodes	Branched nodes
complete enumeration	1,820.84	7,382,230	840,203
	(1,777.74)	(13,553,372)	(1,230,246)
minimal delaying alternatives (Theorem 1)	1,135.81	3,363,698	566,790
	(1,569.14)	(5,866,447)	(827,583)
lb_0	902.22	5,904,531	702,046
	(6,517.54)	(42,218,695)	(5,015,775)
redundant delaying alternatives (Theorem 2)	231.49	1,025,791	183,670
	(1,680.69)	(7,482,414)	(1,319,280)
redundant delaying modes (Theorem 3)	174.28	700,173	132,730
	(1,272.42)	(5,138,468)	(961,071)
lb_3^g (Theorem 4)	152.90	443,680	20,362
	(1,115.68)	(3,246,304)	(577,536)
subset dominance rule (Theorem 5)	60.00	148,201	27,508
	(369.55)	(1,077,582)	(195,539)
solution space reduction (Theorem 6)	0.64	1675	642
	(2.50)	(6,065)	(2,521)

5.2.4. Impact of node fathoming: a full factorial experiment

Although Tables VI and VIII give a clear indication that the node fathoming rules lead to a substantial reduction in computational effort, a detailed analysis is required to assess the power of each of the node fathoming rules independently. Therefore, we set up a full factorial experiment in which all the combinations of each of those rules can be examined. The design of the experiment is given in Table IX. Notice that, in an attempt to reduce the size of the experimental design, Theorems 2 and 3 are not examined separately because of their similar nature. Also the use of *minimal* delaying alternatives and lb_0 is assumed. Separate results are reported for a *complete* enumeration algorithm and for an enumeration algorithm based on *minimal* delaying alternatives only. Remember that a time limit of 1 hour was imposed for the complete enumeration algorithm and the algorithm based on Theorem 1 only.

Table IX. The experimental design

A. Minimal delaying alternatives *(Theorem 1)*	N	Y																	
B. lb_0	N	N	Y																
C. Redundant delaying alternatives and delaying modes *(Theorems 2 and 3)*	N	N	N						Y										
D. lb_3^g *(Theorem 4)*	N	N	N				Y				N				Y				
E. Subset dominance rule *(Theorem 5)*	N	N	N		Y		N		Y		N		Y		N		Y		
F. Solution space reduction *(Theorem 6)*	N	N	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	

The results of the experiment are given in Table X. For each combination of node fathoming rules, the values reported are:

- the average computation time and its standard deviation
- the average number of nodes created in the search tree and its standard deviation
- the average number of nodes actually branched from and its standard deviation

To get a more profound insight in the quality of our procedure as a function of the problem characteristics, three signal-to-noise ratio's are given which assess the quality of our procedure. In order to evaluate the performance of a procedure, the average computation time (number of nodes) as well as its variance have to be investigated. The ideal situation is to have a very low average computation time with a very low variance in computation times. This leads to problems regarding the trade-off between a deterioration in the average performance and an amelioration in the performance variation and vice-versa. Taguchi (see Devor et al., 1992) suggests the concept of a signal-to-noise ratio as a means to evaluate system performance. He defines a quality-loss function which describes how the quality of a system deteriorates as it deviates from its ideal target value. Based on a quadratic quality-loss function (in which the quality deteriorates in a quadratic fashion as the system deviates from the target value), Taguchi suggests the following signal-to-noise ratio for a process for which the ideal value is as small as possible, as is the case with optimal procedures for project scheduling problems:

$S/N = -10\log_{10}\frac{\sum_{i=1}^{n} x^2}{n}$, in which x denotes the process characteristic under study (computation time, number of nodes in the search tree) and n is equal to the number of observations (550).

Table X : Experimental results

Theorem 1	N	Y															
lb_0	N	N	Y														
Theorems 2 and 3	N	N	N								Y						
Theorem 4	N	N	N				Y				N				Y		
Theorem 5	N	N	N		Y		N		Y		N		Y		N		Y
Theorem 6	N	N	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y
CPU-time (seconds)	2,188.54	1,766.16	276.27	89.93	84.34	33.95	139.92	45.03	46.96	18.25	59.25	30.80	29.82	18.47	39.65	18.52	18.37	10.12
	(1,726.26)	(1,733.58)	(3,834.47)	(1,290.66)	(1,088.98)	(381.66)	(1,774.08)	(470.51)	(539.99)	(174.79)	(455.49)	(142.61)	(193.07)	(98.66)	(365.27)	(67.96)	(67.96)	(42.67)
created nodes	4,658,047	3,267,513	1,236,804	371,698	346,263	138,596	529,891	187,002	170,687	74,280	237,270	135,383	110,966	74,200	134,946	76,952	60,909	39,916
	(6,612,913)	(3,881,148)	(16,009,278)	(3,764,745)	(3,766,255)	(1,061,114)	(6,406,915)	(1,090,812)	(1,711,727)	(387,851)	(1,738,012)	(489,805)	(628,229)	(289,400)	(1,064,435)	(254,386)	(373,094)	(131,348)
branched nodes	775,455	702,835	265,154	91,255	82,169	33,740	92,615	36,411	33,234	14,462	57,150	31,763	28,819	18,340	28,381	15,359	13,483	8091
	(864,721)	(786,870)	(3,694,834)	(1,249,840)	(1,070,492)	(360,984)	(1,018,034)	(344,557)	(361,440)	(127,351)	(372,317)	(135,810)	(170,817)	(89,830)	(199,179)	(54,696)	(81,392)	(32,199)
S/N (time)	—	—	-71.69	-62.23	-60.76	-51.66	-65.00	-53.48	-54.67	-44.89	-53.23	-43.27	-45.81	-40.02	-51.30	-36.95	-42.63	-32.83
S/N (created nodes)	—	—	-144.11	-131.55	-131.54	-120.58	-136.16	-120.87	-124.70	-111.92	-124.87	-114.11	-116.09	-109.50	-120.60	-108.48	-111.54	-102.75
S/N (branched nodes)	—	—	-131.37	-121.95	-120.61	-111.18	-120.18	-110.79	-111.19	-102.15	-111.51	-102.88	-104.77	-99.24	-106.06	-95.08	-98.32	-90.42
rank	—	—	16 / 16 / 16	14 / 14 / 15	13 / 13 / 13	9 / 8 / 9	15 / 15 / 14	11 / 10 / 10	12 / 11 / 11	6 / 5 / 5	10 / 12 / 12	5 / 6 / 6	7 / 7 / 7	3 / 3 / 4	8 / 9 / 8	2 / 2 / 2	4 / 4 / 3	1 / 1 / 1
average rank	—	—	16	14	13	9	15	10	11	5	12	6	7	3	8	2	4	1

Maximizing that signal-to-noise ratio is equivalent to minimizing the expected quality loss associated with striving to be on target (computational effort as low as possible) with the smallest variation. The combinations of the design variables that maximize the signal-to-noise ratio are selected as the optimal parameter settings for the process (procedure).

Three such signal-to-noise ratio's are reported, each with respect to a different performance criterion: S/N (time), S/N (created nodes) and S/N (branched nodes). The different problem classes are ranked in the order suggested by the S/N ratio's. The higher the S/N ratio, the higher the quality of the procedure. A signal-to-noise ratio is not given for the two columns that represent the complete enumeration algorithm and the procedure based on Theorem 1 only, since many problem instances could not be solved to optimality within the given time limit. This clearly affects the quality of the procedure but cannot be measured by the S/N ratio's.

The different parameter settings are ranked in the order suggested by the S/N ratio's. From Table X, we can conclude that the optimal parameter settings for our procedure result in the inclusion of all node fathoming rules. When all fathoming rules are included, the average computation time and the number of created nodes in the search tree (and nodes branched from) are minimal, and so are their variances. Consequently, the quality of the full-fledged procedure is highest, reflected by the three signal-to-noise ratio's, which are maximal.

We have established that the inclusion of all proposed node fathoming rules maximizes the quality of our procedure. However, it would be interesting to know which rules are more essential than others, in other words, to determine an order which reflects the impact of the node fathoming rules. Several approaches can be followed to derive such an order. We could, for instance, examine the decrease in quality when a certain node fathoming rule is left out. The results for each of the rules and for each of the three quality measures are given in Table XI. The last column indicates the rank of the proposed node fathoming rules, low ranks associated with a large decrease in quality when the rule is left out.

Table XI. The quality effect of node fathoming rules (simple effect)

Theorems	S/N (CPU time) decrease	S/N (created nodes) decrease	S/N (branched nodes) decrease	Rank decrease	Rank
2 & 3	12.06	9.17	11.73	4	1
4	7.19	6.75	8.82	2	3
5	4.12	5.73	4.66	1	4
6	9.80	8.79	7.90	3	2

Clearly, folowing this approach, Theorems 2 and 3 constitute the most important node fathoming rule, followed by Theorem 6, Theorem 4 and Theorem 5. Another approach to derive an order of importance for the node fathoming rules is to examine the average decrease in quality for each combination of the other node fathoming rules (and not only for the case when all other node fathoming rules are included). The results for this analysis are given in Table XII. The results are quite similar as the ones obtained by the first approach (Table XI). The only difference is that now, Theorem 5 seems more important than Theorem 4.

Table XII. The quality effect of node fathoming rules (main effect)

Theorems	average S/N (CPU time) decrease	average S/N (created nodes) decrease	average S/N (branched nodes) decrease	Rank
2 & 3	14.79	14.19	15.14	1
4	5.87	6.92	8.67	4
5	7.99	9.02	7.74	3
6	9.97	11.23	8.79	2

5.2.5. Impact of node fathoming as a function of problem characteristics

Again, a more detailed analysis is required to reveal how the influence of the inclusion of node fathoming rules depends on the percentage and characteristics of the maximal time lags and the nature of the minimal time lags. Table XIII shows the impact of the node fathoming rules for the problems of class 10. We see that the impact of the node fathoming rules is much higher for this problem class, i.e. when many (tight) maximal time lags are included and when most of the minimal time lags allow for activity overlaps. For instance, the ratio of the required CPU time of the procedure without the node fathoming rules except the use of minimal delaying alternatives and lb_0 versus the required CPU time of the full-fledged procedure for problem class 10 equals 1410 for problem class 10 instead of 27 for all 550 problem instances.

Table XIV indicates the importance of the proposed node fathoming rules, as a function of the decrease in quality on the problems of class 10, when they are left out of the procedure. The results are different from those in Table XI. Theorem 6 now constitues the most important rule, followed by Theorems 2 and 3, Theorem 5 and Theorem 4. Table XV gives the results of the second approach. The fact that Theorem 6 is now very important is very logical, since since the inclusion of many maximal time lags and minimal time lags that allow for activity overlaps increases the effectiveness of this preprocessing rule.

Table XIII : Experimental results for problem class 10

Theorem 1	N	Y															
lb_0	N	N	Y														
Theorems 2 and 3	N	N	N								Y						
Theorem 4	N	N	N				Y				N				Y		
Theorem 5	N	N	N		Y		N		Y		N		Y		N		Y
Theorem 6	N	N	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y
CPU-time (seconds)	1,820.84	1,135.81	902.22	17.15	191.49	3.57	668.10	18.12	159.28	3.75	174.28	1.37	56.15	0.65	152.90	1.35	60.00	0.64
	(1,777.74)	(1,569.14)	(6,517.54)	(104.60)	(1,379.41)	(21.53)	(4,773.45)	(111.21)	(1,139.70)	(22.84)	(1,272.42)	(5.52)	(407.31)	(2.49)	(1,115.68)	(5.60)	(369.55)	(2.50)
created nodes	7,382,230	3,363,698	5,904,531	181.012	1,220,312	36,977	2,819,581	180,280	697,329	36,712	700,173	3,847	222,241	1,888	443,630	3,438	148,201	1723
	(13,553,372)	(5,866,447)	(42,218,695)	(1,164,392)	(8,702,697)	(237,070)	(19,390,434)	(1,164,482)	(4,838,821)	(237,097)	(5,138,468)	(14,553)	(1,622,506)	(6,497)	(3,246,304)	(13,856)	(1,077,582)	(6,065)
branched nodes	840,203	566,790	702,046	18,346	153,592	3,785	326,986	18,076	84,430	3,696	132,730	1,574	43,107	723	20,362	1,408	27,508	640
	(1,230,246)	(827,583)	(5,015,775)	(107,417)	(1,094,535)	(21,993)	(2,246,623)	(107,424)	(585,485)	(21,989)	(961,071)	(6,196)	(309,375)	(2,678)	(577,536)	(5,924)	(195,539)	(2,521)
S/N (time)	—	—	-76.29	-40.43	-62.80	-26.70	-73.58	-40.96	-61.14	-27.21	-62.10	-15.02	-52.20	-8.15	-60.95	-15.14	-51.36	-8.15
S/N (created nodes)	—	—	-152.52	-121.35	-138.80	-107.52	-145.77	-121.35	-133.71	-107.52	-134.22	-83.48	-124.21	-76.53	-130.23	-83.02	-120.65	-75.90
S/N (branched nodes)	—	—	-134.01	-100.67	-120.79	-86.89	-127.04	-100.67	-115.36	-86.89	-119.66	-76.04	-109.82	-68.79	-115.24	-75.62	-105.83	-58.23
rank	—	—	16 / 16 / 16	7 / 8 / 7	14 / 14 / 14	5 / 5 / 5	15 / 15 / 15	8 / 8 / 7	12 / 12 / 12	6 / 5 / 6	13 / 13 / 13	3 / 4 / 4	10 / 10 / 10	1 / 2 / 2	11 / 11 / 11	4 / 3 / 3	9 / 7 / 9	1 / 1 / 1
average rank	—	—	16	7	14	5	15	8	12	6	13	4	10	2	11	3	9	1

Table XIV. The quality effect of node fathoming rules for class 10 (simple effect)

Theorems	S/N (CPU time) decrease	S/N (created nodes) decrease	S/N (branched nodes) decrease	Rank decrease	Rank
2 & 3	9.06	31.62	28.66	5	2
4	0.00	0.63	10.56	1	4
5	6.99	7.12	17.39	2	3
6	43.21	44.75	47.60	8	1

Table XV. The quality effect of node fathoming rules for class 10 (main effect)

Theorems	average S/N (CPU time) decrease	average S/N (created nodes) decrease	average S/N (branched nodes) decrease	Rank
2 & 3	17.01	25.04	17.89	2
4	0.65	2.56	3.97	4
5	10.85	10.89	12.04	3
6	39.83	37.93	36.74	1

5.2.6. Heuristic performance

We also tested the heuristic performance of a truncated version of our branch-and-bound algorithm. Demeulemeester and Herroelen (1995) report that their truncated branch-and-bound procedure for the RCPSP outperforms all known heuristics for the RCPSP when it is allowed to run for a very short time (0.1 seconds). Also when the procedure is truncated after the first feasible solution is found, the solution quality is comparable to the minimum slack heuristic, which is known to rank among the best performing priority based (single-pass) heuristics for the RCPSP (see, for instance, Alvarez-Valdes and Tamarit, 1989; Boctor, 1990; Kolisch, 1994). Table XVI indicates our findings. Several settings for the time limit were examined, namely:

- stop when the first feasible solution is encountered
- search for 1 second
- search for 10 seconds

The reported values are the number of problems solved to (verified) optimality, the number of problems for which the optimal solution was found (but not necessarily verified), the average deviation from the optimal solution, the required computation time, the number of nodes created and branched from in the search tree and the number of problems for which a feasible solution was not found within the given time limit. In each case for which the procedure could not

find a feasible solution within the given time limit, the corresponding problem was resource-infeasible, which we can evaluate as a zero deviation from the optimum (the correct solution was found, but not yet proven). Naturally, this needs not always be the case. Therefore, these problem instances were not taken into account when calculating the average deviation from the optimal solution. Notice that for the first case, a feasible solution is always obtained since that was exactly the stopping criterion for the procedure.

Table XVII shows the results obtained for the problems of class 10 only. Although the heuristic performance already seems very encouraging for all 550 problem instances, it gets even better when many (tight) maximal time lags and minimal time lags that allow for activity overlaps are included. However, that finding the first feasible solution becomes more difficult (higher CPU times and more nodes in the search tree). This is a logical result since the inclusion of maximal time lags (and especially tight ones) reduced the solution space such that it becomes much more difficult to find feasible solutions. Notice, however, that the average quality of the obtained feasible solution (measured by the average deviation from the optimal solution) has improved. When a certain time limit is imposed, the fact that good feasible solutions are harder to find is offset by the fact that the proposed node fathoming rules begin to kick in, which results in the pruning of large portions of the search tree.

Table XVI. Heuristic performance analysis

	(a) first feasible solution	(b) 1 second	(c) 10 seconds
solved to optimality	113	370	474
optimal solution found	270	462	527
average % deviation	4.65 %	0.8 %	0.1 %
CPU time (seconds)	0.08 (0.82)	0.39 (0.45)	2.19 (3.63)
created nodes	213 (2,012)	1,695 (2,250)	10,260 (19,022)
nodes branched from	81 (841)	339 (399)	1,880 (3164)
no feasible solution found	0	6	1

Table XVII. Heuristic performance analysis for problem class 10

	(a) first feasible solution	(b) 1 second	(c) 10 seconds
solved to optimality	17	53	54
optimal solution found	36	53	54
average % deviation	3.75 %	0 %	0 %
CPU time (seconds)	0.44 (2.45)	0.19 (0.31)	0.50 (1.63)
created nodes	1081 (5,994)	604 (970)	1,349 (3,921)
nodes branched from	467 (2,503)	184 (309)	510 (1,689)
no feasible solution found	0	2	1

5.2.5. Relative performance to other optimal and suboptimal procedures

As mentioned before, the only other optimal procedure for the RCPSP-GPR described in the literature is the branch-and-bound procedure of Bartusch et al. (1988). However, a comparative study of the effectiveness and efficiency of this procedure and our branch-and-bound algorithm is impossible since the code of the algorithm of Bartusch et al. (1988) is lost (Möhring 1996). Little computational experience with the algorithm of Bartusch et al. (1988) has been reported. Heuristics for the RCPSP-GPR have been presented by Brinkmann and Neumann (1994), Zhan (1994) and Neumann and Zhan (1996). However, these algorithms are also no longer available (Neumann 1995). New and improved versions of these heuristics are being developed (Neumann 1995). Therefore, a study of the relative performance of our procedure versus other optimal and suboptimal procedures has to be left for future research.

6. Conclusions

In this paper we present a solution procedure for the resource-constrained project scheduling problem with generalized precedence relations (RCPSP-GPR) with the objective of minimizing the makespan of the project. The RCPSP-GPR extends the RCPSP to arbitrary minimal and maximal time lags between the starting and completion times of activities. The procedure is a depth-first branch-and-bound algorithm in which the nodes in the search tree represent the original project network extended with precedence relations to resolve a resource conflict in the parent node. Resource conflicts are resolved using the concept of minimal delaying alternatives. Precedence- and resource-based lower bounds as well as dominance rules are used to fathom large portions of the search tree. The procedure can be readily extended to other regular measures of performance through some minor modifications. Even nonregular measures of performance, such as the maximization of the net present value of the project, can be handled.

The procedure has been programmed in Visual C^{++} for use on a personal computer. Extensive computational experience is obtained, which indicates that all the proposed node fathoming rules lead to a significant reduction in the required computation time, the number of nodes created and the number of nodes branched from in the search tree. Also the use of a truncated version of the procedure as a heuristic for the RCPSP-GPR yields encouraging results.

Schwindt (1995) is working on a procedure, ProGen/max, which generates RCPSP-GPR instances with preset values for several control parameters which are considered to be good measures of complexity for the RCPSP-GPR. Using this procedure, we would be able to perform a more extensive experiment in which the impact of those complexity measures on the required computational effort can be examined. However, the code for the generator is not yet completed. Therefore, this will have to be reserved for future research.

Appendix: Proofs

PROOF OF THEOREM 1.

The proof consists of three parts. We will prove that (a) the delaying strategy based on (not necessarily *minimal*) delaying alternatives leads to the optimal solution, (b) it is sufficient to consider only *minimal* delaying alternatives, and (c) the procedure will find the optimal solution in a finite number of steps.

Lemma 1. The delaying strategy based on delaying alternatives leads to the optimal solution

PROOF. Consider the project network associated with the *root node (node 0)* of the search tree (i.e., the original project network). If this network is time-infeasible, no feasible solution can be obtained. If the earliest start schedule for this network has no resource conflict, it is optimal. Therefore, we assume that the project network in the root node of the search tree is time-feasible but resource-infeasible. Suppose that $S(t^*) = \{i_1, i_2, \ldots, i_p\}$ is the conflict set of activities in progress in period $]t^*\text{-}1, t^*]$, the period in which the first resource conflict occurs. We can now use the following Lemma:

Lemma 1.1. In each feasible solution that may result from resolving a resource conflict created by the conflict set $S(t^*)$, the precedence relation $i_k \prec i_l$ must be satisfied for at least one pair of activities $(k,l) \in S(t^*)$.

PROOF. See Lemma 3.6 in Bartusch et al. (1988).

Therefore, resolving a resource conflict at node 0 by branching into $p(p-1)$ nodes, each of which adds a different precedence constraint $i_k \prec i_l$ $(k,l) \in S(t^*)$, guarantees that each feasible solution that could be obtained at node 0 before the branching, can still be obtained. More formally: $\Omega_0 = \bigcup_{k=1}^{p(p-1)} \Omega_k$, in which Ω_k represents the set of feasible solutions that can be obtained when branching from node k. Repeating this branching strategy throughout the search tree will lead to the optimal solution.

Our delaying strategy, however, is based on delaying alternatives and delaying modes. Each delaying mode M_m for which the delaying alternative D_d consists of a single activity, corresponds to a precedence relation as specified in Lemma 1.1. It remains to be shown that a precedence constraint $\{i_k \prec i_l\}$ imposed by Lemma 1.1, which would not be identified by our procedure as a possible delaying mode, is dominated and can be ommitted. The reason for our procedure not to generate the delaying mode $\{i_k \prec i_l\}$ can only be that conflict activity i_l does not release enough resources to resolve the resource conflict in period $]t^*\text{-}1, t^*]$. In other words, adding

a constraint $\{i_k \prec i_l\}$ at level 1 of the search tree is not enough to resolve the resource conflict in node 0, and, using Lemma 1.1, it would be necessary to delay another activity $i_n \neq i_l \in S(t^*)$ by another activity $i_m \notin \{i_l, i_n\}$: $\{i_m \prec i_n\}$ at the second level of the search tree.

Case 1: delaying activity i_n releases enough resources by itself to resolve the resource conflict

In this case, the constraint $\{i_m \prec i_n\}$ would be identified as a delaying mode by our procedure at level 1 of the search tree. Therefore, the set of feasible solutions that can be obtained by branching from the node on the second level of the search tree is a subset of the set of feasible solutions that can be obtained with our procedure when branching from the node $\{i_m \prec i_n\}$ at level 1 of the search tree.

Case 2: delaying activity i_n does not releases enough resources by itself to resolve the conflict

Case 2-1: $i_m \neq i_k$

In this case, we delay i_l by i_k and i_n by i_m ($i_l \neq i_n$ and $i_k \neq i_m$).

Case 2-1-1: the resource conflict is resolved

If the resource conflict is resolved by adding the delaying modes $\{i_k \prec i_l\}$ and $\{i_m \prec i_n\}$ to the project network, all corresponding feasible solutions will also be obtained by the delaying mode $\{i_k \prec i_l, i_k \prec i_n\}$ or by delaying mode $\{i_m \prec i_l, i_m \prec i_n\}$. Suppose that in a feasible solution resulting from resolving the resource conflict, activity i_k finishes before activity i_m (or at the same time). Then this feasible solution will not be eliminated by relaxing the constraint $\{i_m \prec i_n\}$ by $\{i_k \prec i_n\}$. If, on the other hand, activity i_m finishes before activity i_k, then relaxing the constraint $\{i_k \prec i_l\}$ by $\{i_m \prec i_l\}$ will not eliminate this feasible solution. Moreover, the two relaxed problems consisting of delaying modes $\{i_k \prec i_l, i_k \prec i_n\}$ and $\{i_m \prec i_l, i_m \prec i_n\}$ will be identified by our procedure, since $\{i_l, i_n\}$ is a valid delaying alternative. Therefore, all the delaying modes $\{i_k \prec i_l, i_m \prec i_n\}$ imposed by Lemma 1.1 are dominated by the two delaying modes $\{i_k \prec i_l, i_k \prec i_n\}$ and $\{i_m \prec i_l, i_m \prec i_n\}$ in our search procedure.

Case 2-1-2: the resource conflict is not resolved

As the resource conflict is not resolved by the delaying modes $\{i_k \prec i_l\}$ and $\{i_m \prec i_n\}$, new delaying modes will have to be added. Eventually, at a certain level of the search tree, a feasible solution will be obtained which was reached by adding a set of extra precedence relations $\left\{\{i_{k_1} \prec i_{l_1}\}, \{i_{k_2} \prec i_{l_2}\}, \ldots, \{i_{k_q} \prec i_{l_q}\}\right\}$, for which the delaying activities are not one and the same activity (at least two have to be different, since at level one and two of the search tree, the

delaying activities were different from one another : $i_m \neq i_k$). Then relaxing the constraints $\{i_{k_x} \prec i_{l_x}\}$ by $\{i_K \prec i_{l_x}\}$, in which i_K is the earliest finishing delaying activity in the feasible solution, will not eliminate the feasible solution attained. The corresponding delaying mode $\{\{i_K \prec i_{l_1}\}, \{i_K \prec i_{l_2}\}, \ldots, \{i_K \prec i_{l_q}\}\}$ will also be identified by our procedure on the first level of the search tree, since $\{i_{l_1}, i_{l_2}, \ldots, i_{l_q}\}$ constitutes a valid delaying alternative. Therefore, the delaying modes $\{\{i_{k_1} \prec i_{l_1}\}, \{i_{k_2} \prec i_{l_2}\}, \ldots, \{i_{k_q} \prec i_{l_q}\}\}$ imposed by Lemma 1.1 are dominated by our delaying modes $\{\{i_K \prec i_{l_1}\}, \{i_K \prec i_{l_2}\}, \ldots, \{i_K \prec i_{l_q}\}\}$ on the first level of the search tree.

Case 2-2: $i_m = i_k$

In this case, we would have delayed i_l and i_n by i_k ($i_l \neq i_n$). If the corresponding earliest start schedule is feasible, the corresponding delaying modes will be identified by our procedure, since $\{i_l, i_n\}$ then constitues a valid delaying alternative and i_k a valid delaying activity. If, however, the earliest start schedule would still not be time-feasible, other precedence constraints would have to be added. Now we would run into Case 2-1 or 2-2, depending on the delaying activity, but now one level down in the search tree. As we have shown in both cases, the corresponding delaying modes will either also be identified by our procedure, or be dominated by a delaying mode generated by our procedure.

So, we have shown that all the delaying modes imposed by Lemma 1.1 in order to eliminate the first resource conflict at the source node 0 will either also be identified or will be dominated by a delaying mode on level 1 of the search tree. Repeating a similar argument for any node created in the search tree, leads to the proof of Lemma 1.

Lemma 2. In order to resolve a resource conflict, it is sufficient to consider minimal delaying alternatives

PROOF. According to Lemma 1, a branching strategy based on delaying alternatives leads to the optimal solution. Lemma 1 does not exclude delaying alternatives D_d that contain other delaying alternatives $D_{d'}$ as a subset. These non-minimal delaying alternatives D_d, however, need not be examined, since the set of feasible solutions we can obtain by branching from a node with a corresponding non-minimal delaying mode will be a proper subset of the set of feasible solutions obtained when branching from a node with a delaying mode corresponding to $D_{d'}$. When we branch from D_d, we obtain the project network created by delaying alternative $D_{d'}$, extended with one or more precedence relations. Therefore, the set of feasible solutions that can be obtained from

node D_d is a proper subset of the set of feasible solutions that can be obtained from node $D_{d'}$. Therefore, delaying alternative D_d (and all corresponding delaying modes) can be eliminated.

Lemma 3. The delaying strategy based on Lemma 2 will lead to the optimal solution in a finite number of steps

PROOF. At each branch of the search tree, we create a number of nodes equal to the number of minimal delaying modes. Clearly, the maximal number of activities in $S(t^*)$ is equal to n. Then, the maximal number of minimal delaying modes is equal to $n!$. This can be seen clearly as follows: If the number of activities in each delaying alternative is equal to 1, the number of delaying alternatives is equal to n and the number of delaying modes equal to $n(n\text{-}1)$ because there are n-1 possible delaying activities for each delaying alternative. If the number of activities per delaying alernative is equal to 2, the maximal number of delaying modes would be equal to $\frac{n(n-1)(n-2)}{2}$, because there are $\frac{n!}{(n-2)!2!}$ delaying alternatives and n-2 possible delaying activities. In general, x activities per delaying alternative would give rise to maximally $\frac{n!}{(n-x-1)!x!}$ delaying modes. Clearly, the maximal number of minimal delaying modes is always smaller than $n!$, even if the number of activities varies from delaying alternative to delaying alternative. Thus, the maximal number of nodes generated during each branching step equals $n!$. We know that the maximal number of zero-lag finish-start precedence relations that we can add to a project network (without affecting time-feasibility) equals $n(n\text{-}1)/2$. Therefore, the maximal number of levels in the search tree equals $n(n\text{-}1)/2$. Thus, the maximal number of nodes generated in the search tree equals $\sum_{i=0}^{n(n-1)/2} (n!)^i$, the maximal number of leaf nodes being equals to $(n!)^{n(n-1)/2}$. According to Lemma 2, one of these nodes is bound to contain the optimal solution to the problem. Since the number of nodes in the search tree is finite, the optimal solution can be found in a finite number of steps.

PROOF OF THEOREM 4.

If we split up each activity i of an RCPSP-GPR instance into unit-duration *subactivities* $(i_1, i_2, \ldots, i_p)$, connected with zero-lag minimal finish-start time lags, we obtain the preemptive version of the RCPSP-GPR (see also Demeulemeester and Herroelen, 1996b for the preemptive RCPSP), provided that the precedence relations between the activities are represented correctly, i.e. connected to the correct subactivity. However, if we also add zero-lag *maximal* finish-start time lags, we again obtain the original problem, since all the subactivities of a given activity have to be performed consecutively. If the project network is represented in its standardized form (as a constraint digraph), all precedence relations are of the start-start type which can be represented in the unit-duration RCPSP-GPR by precedence relations between the first subactivities of each

activity only ($i \prec j \Rightarrow s_{i_1} + d_i \leq s_{j_1}$). The zero-lag minimal and maximal finish-start precedence relations between the subactivities are then represented by two minimal start-start relations equal to 1 and -1 respectively for each consecutive pair of subactivities ($s_{i_k} + 1 \leq s_{i_{k+1}}$ and $s_{i_{k+1}} - 1 \leq s_{i_k}$). As an example, the RCPSP-GPR instance with only two activities from Fig. A-1 can be transformed to the unit-duration RCPSP-GPR given in Fig. A-2.

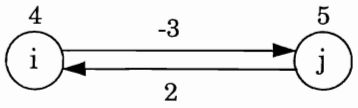

Figure A-1. An RCPSP-GPR example

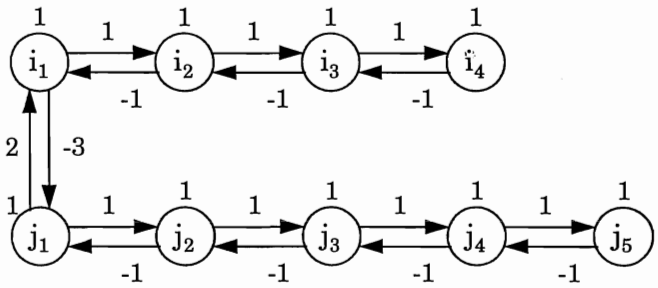

Figure A-2. The corresponding unit-duration RCPSP-GPR

Since the two problems are identical, a lower bound for the unit-duration problem will also be a valid lower bound for the original problem. Now, we will show that the exact value of lb_3^g will be obtained by calculating lb_3' (Demeulemeester and Herroelen 1995) for the unit-duration problem, with the additional assumptions that (a) the order in which the subactivities are placed in the list L are determined by looking at the original problem (i.e. the number of companions is calculated by looking at the original activities, not the subactivities) and (b) (which logically follows from (a)) if a subactivity i_k of activity i is removed from L, so are all the other subactivities $i_l (l \neq k)$ of activity i (which can never be companions of i_k).

Since all the time lags between the subactivities of one and the same activity are equal to 1 (from i_k to i_{k+1}) and -1 (from i_{k+1} to i_k), we know that i_k $(1 \leq k \leq p)$ is a companion of j_l $(1 \leq l \leq q)$ if both $d_{ij} - (k-1) + (l-1) \leq 0$ and $d_{ji} + (k-1) - (l-1) \leq 0$ [A - 1]

i.e. the maximal distance between i_k and j_l or between j_l and i_k may never exceed zero, because then they can never be scheduled in parallel. For the example given in Fig. A-2, activity j_l is a companion of activity i_1 if $-3 - (1-1) + (l-1) \leq 0$ and $2 + (1-1) - (l-1) \leq 0 \Leftrightarrow l \leq 4$ and $l \geq 3$. Therefore, only subactivities j_3 and j_4 are companions of i_1.

Assume we compute lb_3' for the unit-duration RCPSP-GPR. When we remove subactivity i_k from L, we also have to remove its companions from L, i.e. each j_l for which condition [A - 1] applies. By simplifying Eq. A-1, we obtain: remove j_l if $k + d_{ji} \leq l \leq k - d_{ij}$ [A - 2]

(note that $k + d_{ji} \leq k - d_{ij}$ because otherwise: $k + d_{ji} > k - d_{ij} \Rightarrow d_{ij} + d_{ji} > 0 \Rightarrow d_{ii} > 0$, which would result in a time-infeasibility). Again, for the example, when removing subactivity i_1 from L, we also have to remove its companions j_3 and j_4, since $1+2 \leq 3 \leq 1+3$ and $1+2 \leq 4 \leq 1+3$.

Consequently, when we remove all the subactivities i_k from L, we have to remove each subactivity j_l for which condition [A - 2] applies for any subactivity i_k. For the problem example, when removing subactivity i_2, we have to remove its companions j_4 (which was already removed) and j_5; when removing subactivity i_3, we have to remove j_5 (which was already removed) and when removing subactivity i_4, there is no need to remove any subactivity. The smallest value l for which [A - 2] is true is equal to $\max\{1, 1+d_{ji}\}$ (for the example: 3), whereas the largest value for l equals $\min\{d_j, d_i - d_{ij}\}$ (for the example: 5). Therefore, the number of subactivities from activity j to be removed equals: $\min\{d_j, d_i - d_{ij}\} - \max\{1, 1+d_{ji}\} + 1$. However, the maximal number of subactivities of activity j to be removed can never exceed d_i. Therefore, the number of subactivities j_l to be removed from L if all subactivities i_k are removed from L equals: $\min\{\min\{d_j, d_i - d_{ij}\} - \max\{1, 1+d_{ji}\} + 1, d_i\}$ [A - 3]

For our problem example, the number of subactivities j_l to be removed from L equals: $\min\{\min\{5, 4+3\} - \max\{1, 1+2\} + 1, 4\} = 3$.

We will now examine each of the combinations given in Table I which represent time-feasible project networks and in which activities i and j are companions.

Case 1: $d_{ij} \leq 0$ and $0 < d_{ji} < d_j$ *(row 2 and columns 3-4 in Table I)* [A - 4]

Using Eq. A-4, Eq. A-3 can be simplified to: $\min\{\min\{d_j, d_i - d_{ij}\} - 1 - d_{ji} + 1, d_i\} =$ $\min\{\min\{d_j, d_i - d_{ij}\} - d_{ji}, d_i\} = \min\{\min\{d_j - d_{ji}, d_i - d_{ij} - d_{ji}\}, d_i\} =$ $\min\{d_j - d_{ji}, d_i - d_{ij} - d_{ji}, d_i\}$. We know that $d_i \leq d_i - d_{ij} - d_{ji}$, because otherwise we would get $d_{ij} + d_{ji} > 0$ which leads to time-infeasibility. Therefore, the number of subactivities of activity j to be removed equals $\min\{d_j - d_{ji}, d_i\}$, which is equivalent to a reduction of the remaining

duration of activity j with the same amount. This value corresponds to the values of cells (2,3) and (2,4) in Table I. Notice that because $d_{ji} < d_j$ this value can never become negative.

Case 2: $0 < d_{ij} < d_i$ and $d_{ji} \leq 0$ *(column 2 and rows 3-4 in Table I)* [A - 5]

Using Eq. A-5, Eq. A-3 can be simplified to: $\min\{\min\{d_j, d_i - d_{ij}\} - 1 + 1, d_i\} = \min\{\min\{d_j, d_i - d_{ij}\}, d_i\} = \min\{d_j, d_i - d_{ij}, d_i\} = \min\{d_j, d_i - d_{ij}\}$, which is the value for cells (3,2) and (4,2) in Table I. Notice again that this value can never become negative. Notice also that the value given in Table I ($d_i - d_{ij}$) is slightly different, because no check is needed that the remaining duration of activity j becomes negative (if $d_j^r \leq 0$, it is completely removed from L).

Case 3: $d_{ij} \leq 0$ and $d_{ji} \leq 0$ *(rows 3-4 and columns 3-4 in Table I)* [A - 6]

Using Eq. A-6, Eq. A-3 can be simplified to: $\min\{\min\{d_j, d_i - d_{ij}\} - 1 + 1, d_i\} = \min\{\min\{d_j, d_i - d_{ij}\}, d_i\} = \min\{d_j, d_i - d_{ij}, d_i\} = \min\{d_j, d_i\}$, which is the value for cells (3,3), (3,4), (4,3) and (4,4) in Table I. Notice again the slight difference because of the nonnegativity constraint of the remaining duration of activity j, which is not needed in our procedure.

For the sake of completeness, we will extend the proof to the combinations of time lags for which activities i and j are no companions (cells (4,1) and (1,4) in Table I). A similar reasoning can be given for the time lag combinations which are not time-feasible (cells (1,1), (1,2), (2,1), (2,2), (3,1) and (1,3) in Table I).

Case 4: $d_{ij} \leq -d_j$ *and* $d_{ji} \geq d_j$ *(row 1, column 4 in Table I)* [A - 7]

Using Eq. A-7, Eq. A-3 can be simplified to: $\min\{\min\{d_j, d_i - d_{ij}\} - 1 - d_{ji} + 1, d_i\} = \min\{\min\{d_j, d_i - d_{ij}\} - d_{ji}, d_i\} = \min\{d_j - d_{ji}, d_i - d_{ij} - d_{ji}, d_i\}$. Because $d_{ji} \geq d_j$, we know that $d_j - d_{ji} \leq 0$. Therefore, the minimum will be smaller than zero, which is a logical result because then activities i and j will not be companions at all (as is given in cell (1,4) in Table I). Subsequently, no subactivities j_l have to be removed from L, or, in the original problem, the remaining duration of activity j in L need not be reduced.

Case 5: $d_{ij} \geq -d_i$ *and* $d_{ji} \leq -d_j$ *(row 1, column 4 in Table I)*

Similar argument as for Case 4. □

PROOF OF THEOREM 5.

If the set of added precedence constraints which leads to the project network in node x contains as a subset another set of precedence constraints leading to the project network in a previously examined node y, the project network obtained in node x consists of the project network obtained in node y, extended with zero or more extra zero-lag finish-start precedence relations. Therefore, since the problem in node x is more constrained than the problem in node y, the set of feasible solutions which can be obtained when branching from node x is a subset of the set of feasible solutions which can be obtained when branching from node y ($\Omega_x \subset \Omega_y$). Therefore, the best possible solution which can be obtained when branching from node x can never be superior to the best possible solution that can be obtained when branching from node y (they can be equally good since the same solution can be present in both solution sets).

Because node y stems from another part of the search tree than node x, we know that node y is not a parent node of node x. Node y is already examined and, because of the nature of the depth-first (backtracking) search procedure, node y is also backtracked upon, because otherwise we can not reach a node x *in another part of the search tree*. Therefore, we know that the best possible solution that can be obtained when branching from node y is already determined. Therefore, node x can be fathomed since no superior solution can be obtained when branching from it. □

PROOF OF THEOREM 6.

We know that $\exists\, i, j \in V$ and resource type k for which $r_{ik} + r_{jk} > a_k$ [A - 8]

and $-d_j < d_{ij} < d_i$. [A - 9]

Suppose that there exists a feasible solution for the RCPSP-GPR. In each feasible solution (therefore also in the optimal solution), Eq. A-8 guarantees that either $i \prec j$ or $j \prec i$. Suppose that $j \prec i \;\Rightarrow\; s_j + d_j \le s_i$. [A - 10]

We can derive from Eq. A-9 that $d_{ij} > -d_j$. [A - 11]

Combining the general expression $s_i + d_{ij} \le s_j$ with Eq. A-11, we get $s_i - d_j < s_j$. [A - 12]

Substituting [A - 10] in [A - 12] yields: $s_j + d_j - d_j < s_j \;\Rightarrow\; s_j < s_j$, which is impossible. Therefore, in each feasible solution: $i \prec j \;\Rightarrow\; s_i + d_i \le s_j$. Consequently, we can set $l_{ij} = d_i$.

Now suppose that there does not exist a feasible solution for the RCPSP-GPR. Then, adding the constraint $l_{ij} = d_i$ will not change the fact that no feasible solution is obtained, since adding this precedence constraint will only constrain the problem more, leading to a set of feasible solutions which is a subset of the original set. Since the original set was empty, so will the new set of feasible solutions. □

References

Ahuja, R.K., Magnanti, T.L. and Orlin, J.B., 1989, "Network flows", in: Nemhauser, G.L., Rinnooy Kan, A.H.G. and Todd, M.J. (Eds.), *Handbooks in Operations Research and Management Science*, Elsevier, Amsterdam, 258-263.

Alvarez-Valdés, R. and Tamarit, J.M., 1989, "Heuristic algorithms for resource-constrained project scheduling", in: Slowinski, R. and Weglarz, J. (Eds.), *Advances in Project Scheduling*, Elsevier, Amsterdam, 134-143.

Bartusch, M., Möhring, R.H. and Radermacher, F.J., 1988, "Scheduling project networks with resource constraints and time windows", *Annals of Operations Research*, 16, 201-240.

Bell, C.E. and Park, K., 1990, "Solving resource-constrained project scheduling problems by A* search", *Naval Research Logistics Quarterly*, 37, 61-84.

Boctor, F.F., 1990, "Some efficient multi-heuristic procedures for resource-constrained project scheduling", *European Journal of Operational Research*, 49, 3-13.

Brinkmann, K. and Neumann, K., 1994, "Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags: the minimum project-duration and resource-levelling problem", Technical Report WIOR-443, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.

Crandall, K.C., 1973, "Project planning with precedence lead / lag factors", *Project Management Quarterly*, 4, 18-27.

Davis, E.W., 1973, "Project scheduling under resource constraints - an historical review and categorization of procedures", *AIIE Transactions*, 5 (4), 297-313.

Demeulemeester, E., 1992, *Optimal Algorithms for Various Classes of Multiple Resource-Constrained Project Scheduling Problems*, Ph.D. Dissertation, Department of Applied Economics, Katholieke Universiteit Leuven.

Demeulemeester, E. and Herroelen, W., 1992, "A branch-and-bound procedure for the multiple resource-constrained project scheduling problem", *Management Science*, 38, 1803-1818.

Demeulemeester, E. and Herroelen, W., 1995, "New benchmark results for the resource-constrained project scheduling problem", Research Report 9521, Department of Applied Economics, Katholieke Universiteit Leuven.

Demeulemeester, E. and Herroelen, W., 1996a, "A branch-and-bound procedure for the generalized resource-constrained project scheduling problem", *Operations Research*, to appear.

Demeulemeester, E. and Herroelen, W., 1996b, "A branch-and-bound procedure for the preemptive resource-constrained project scheduling problem", *European Journal of Operational Research*, to appear.

De Reyck, B., 1995a, "Project scheduling under generalized precedence relations - a review: part 1", Research Report 9517, Department of Applied Economics, Katholieke Universiteit Leuven.

De Reyck, B., 1995b, "Project scheduling under generalized precedence relations - a review: part 2", Research Report 9518, Department of Applied Economics, Katholieke Universiteit Leuven.

Devor, R.E., Chang, T. and Sutherland, J.W., 1992, *Statistical Quality Design and Control, Contemporary Concepts and Methods*, Macmillan, New York.

Elmaghraby, S.E., 1977, *Activity Networks: Project Planning and Control by Network Models*, Wiley, New York.

Elmaghraby, S.E. and Herroelen, W., 1990, "The scheduling of activities to maximize the net present value of projects", *European Journal of Operational Research,* 49, 35-49.

Elmaghraby, S.E. and Kamburowski, J., 1992, "The analysis of activity networks under generalized precedence relations", *Management Science*, 38, 1245-1263.

Grinold, R.C., 1972, "The payment scheduling problem", *Naval Research Logistics Quarterly*, 19, 123-136.

Herroelen, W. and Gallens, E., 1993, "Computational experience with an optimal procedure for the scheduling of activities to maximize the net present value of projects", *European Journal of Operational Research*, 65, 274-277.

Herroelen, W., Demeulemeester, E. and Van Dommelen, P., 1996, "An optimal recursive search procedure for the deterministic unconstrained max-npv project scheduling problem", Research Report, Department of Applied Economics, Katholieke Universiteit Leuven.

Icmeli, O and Erengüç, S.S., 1996, "A branch-and-bound procedure for the resource-constrained project scheduling problem with discounted cash flows", *Management Science*, to appear.

Jensen, P.A. and Barnes, J.W., 1987, *Network Flow Programming*, Robert E. Krieger Publishing Company, Florida.

Kelley, J.E., Jr. and Walker, M.R., 1959, "Critical path planning and scheduling", *Proceedings Eastern Joint Computing Conference*, 16, 160-172.

Kerbosh, J.A.G.M. and Schell, H.J., 1975, "Network planning by the Extended METRA Potential Method", Report KS-1.1, University of Technology Eindhoven, Department of Industrial Engineering.

Kolisch, R., 1994, *Project scheduling under resource constraints: efficient heuristics for several problem classes*, Ph.D. dissertation, Institut für Betriebwirtshaftslehre, Christian-Albrechts-Universität zu Kiel.

Kolisch, R., Sprecher, A. and Drexl, A., 1992, "Characterization and generation of a general class of resource-constrained project scheduling problems: Easy and hard instances", *Research* report 301 , Institut für Betriebwirtshaftslehre, Christian-Albrechts-Universität zu Kiel.

Lawler, E.L., 1976, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.

Malcolm, D.G., Roseboom, J.H., Clark, C.E. and Fazar, W., 1959, "Applications of a technique for R&D program evaluation (PERT)", *Operations Research*, 7 (5), 646-669.

Mingozzi, A., Maniezzo, V., Ricciardelli, S. and Bianco, L., 1994, "An exact algorithm for project scheduling with resource constraints based on a new mathematical programming formulation", Technical Report 32, Department of Mathematics, University of Bologna.

Moder, J.J., Phillips, C.R. and Davis, E.W., 1983, *Project management with CPM, PERT and precedence diagramming*, Van Nostrand Reinhold Company, Third Edition.

Möhring, R.H., 1996, *private communication*.

Neumann, K., 1995, *private communication*.

Neumann, K. and Schwindt, C., 1995, "Projects with minimal and maximal time lags: construction of activity-on-node networks and applications", Technical Report WIOR-447, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.

Neumann, K. and Zhan, J., 1996, "Heuristics for the minimum project-duration problem with minimal and maximal time lags under fixed resource constraints", *Journal of Intelligent Manufacturing*, to appear.

Patterson, J.H., 1984, "A comparison of exact procedures for solving the multiple resource-constrained project scheduling problem", *Management Science*, 30 (7), 854-867.

Roy, B., 1962, "Graphes et ordonnancement", *Revue Française de Recherche Opérationelle*, 323-333.

Russell, A.H., 1970, "Cash flows in networks", *Management Science*, 16, 357-373.

Schwindt, C., 1995, "ProGen/max: a new problem generator for different resource-constrained project scheduling problems with minimal and maximal time lags", Technical Report WIOR-449, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe.

Stinson, J.P., Davis, E.W. and Khumawala, B.M., 1978, "Mutliple resource-constrained scheduling using branch-and-bound", *AIIE Transactions* 10 (3), 252-259.

Wiest, J.D., 1981, "Precedence diagramming methods: some unusual characteristics and their implications for project managers", *Journal of Operations Management*, 1, 121-130.

Wikum, E.D., Donna, C.L. and Nemhauser, G.L., 1994, "One-machine generalized precedence constrained scheduling problems", *Operations Research Letters*, 16, 87-99.

Zhan, J., 1994, "Heuristics for scheduling resource-constrained projects in MPM networks", *European Journal of Operational Research*, 76, 192-205.

Figure Captions

Table Captions